Creating Packages with The Luggage

MacTech Magazine May • 2010

MACTECH®

The Journal of Apple Technology

THE iPAD PLATFORM: RIDE THE WAVE

Calling All CPUs: Operation Cues & Central Dispatch

Source Code Metrics

MACTECH.COM

$8.95 US, $12.95 Canada

ISSN 1067-8360 Printed in U.S.A.

RESTRICTED Nominal Ledger
HansaWorld Demo
Last Reg Date 15/9/2006
Values in GBP

ACCOUNT 100 Sales - Audio

Trans No	Trans. Date	Text
		Fwd balance
S/INV.2009001	11/1/2009	001 Against All
S/INV.2009004	4/4/2009	016 London Pride
S/INV.2009002	2/9/2009	019 Pink Snack B
S/INV.2009003	2/11/2009	001 Against All
S/INV.2009005	2/11/2009	001 Against All
		Total
		Fwd balance
		Net Change
		Balance

ACCOUNT 110 Sales - Video

Trans No	Trans. Date	Text
		Fwd balance
S/INV.2009001	11/1/2009	001 Against All

HansaWorld USA, Inc. 1501 Broadway #12067 & 12068, 12th Floor, New York, NY 10036-5601, USA, Phone: +1 646-459-0603, Email: usa@hansaworld.com

Accounting | Reporting | Management

Whether you work alone or lead a team of four hundred people, our solutions will let you bring your business to the Mac.

Integrated CRM | Real-time Mobile | Real-time Consolidation | Integrated Webshop | Built-in Document Management | Powerful Analysis

Books by HansaWorld

Enterprise by HansaWorld

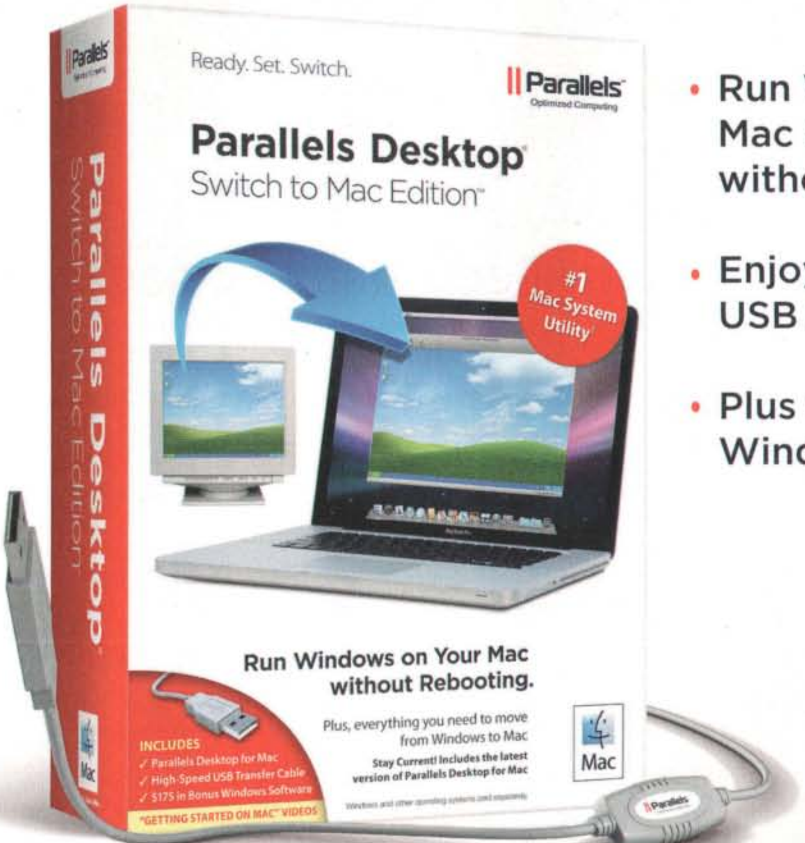
NEW

Switch to Mac has never been easier.

Parallels Desktop® Switch to Mac Edition

Ready. Set. Switch.

- Complete Moving Suite
- Interactive Video Tutorials
- Step-by-Step Easy Migration



Ready. Set. Switch.
Parallels Optimized Computing
Parallels Desktop
Switch to Mac Edition
#1 Mac System Utility
Run Windows on Your Mac without Rebooting.
Plus, everything you need to move from Windows to Mac
Stay Current! Includes the latest version of Parallels Desktop for Mac
INCLUDES
✓ Parallels Desktop for Mac
✓ High-Speed USB Transfer Cable
✓ $175 in Bonus Windows Software
"GETTING STARTED ON MAC" VIDEOS
Mac

- Run Windows and Mac Side-by-Side without Rebooting
- Enjoy Your Favorite USB Devices
- Plus $175 Bonus Windows Software

Parallels Desktop Switch to Mac Edition lets you move programs, documents, media and more right from your PC to your new Mac. Then enjoy the best of both worlds and run Windows and Mac OS X side by side.

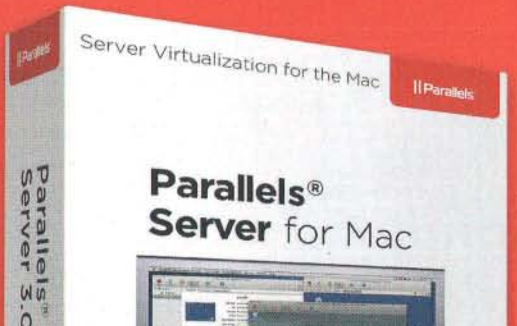
Learn more at
www.parallels.com/products/desktop/stm

Parallels™ Optimized Computing

Phone +1 (425) 282-6448

|| Parallels™

Parallels® Server for Mac


Server Virtualization for the Mac
Parallels® Server for Mac
Parallels® Server 3.0 for Mac
Macworld BEST OF SHOW 2008
Mac
www.parallels.com

Make your Mac Server go farther.

Run any OS you choose.

Run any application on the Apple Xserve today.

The world's first server virtualization solution for the Mac platform.

Reduce Server Count by 84%

Save 60% on IT Budget

Oregon City School District

Performance

Full scale hypervisor solution with bare metal architecture.

Scalability

Virtualize Mac OS X Server, Windows, Linux and more with support for 32- and 64-bit platform and guest OS support.

Flexibility

Hardware-ready for seamless integration into existing IT infrastructures with built-in VM management and maintenance tools.

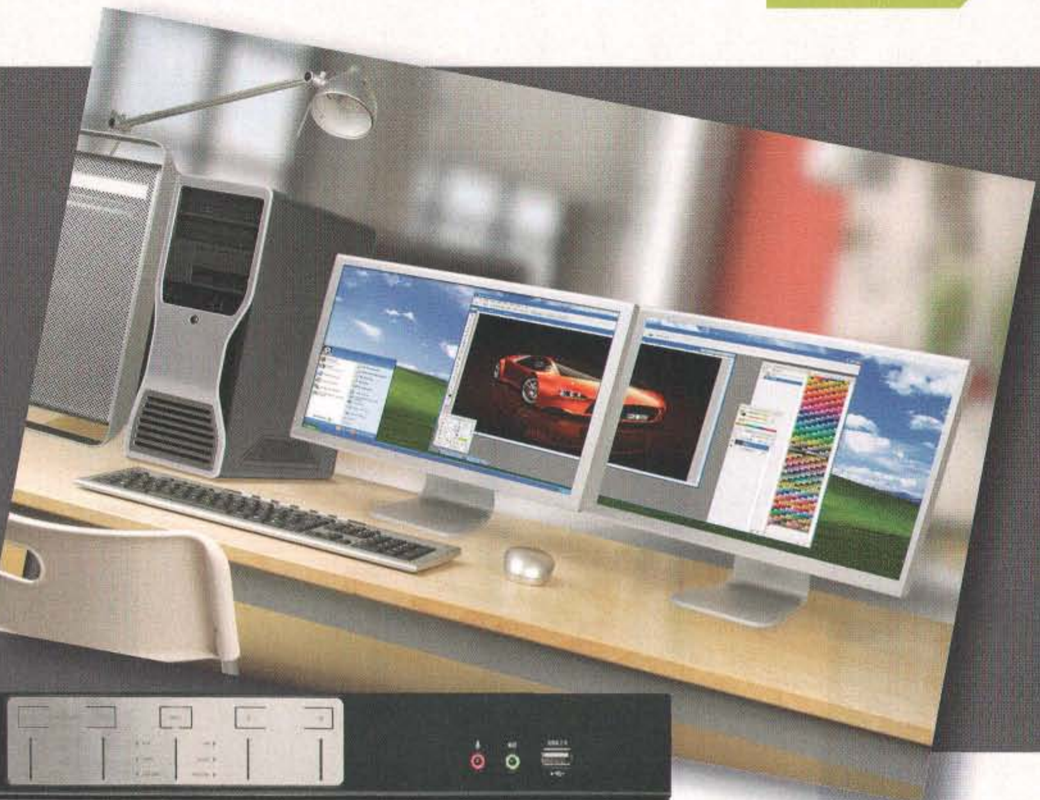
Buy now at www.parallels.com or email offer@parallels.com

Experience the Power of Control

IOGEAR

IOGEAR's 4-Port Dual View DVI KVMP Switch marks a leap in KVM functionality and control by combining IOGEAR's Display Emulation Technology with the widescreen graphics resolution of today's 30+ inch monitors, supporting Dual View / Dual-Link DVI, 2560x1600 high resolution.

Four Dual View / Dual-Link computers can be accessed and controlled from a single console running dual-displays!



model: GCS1644*

4-Port Dual View Dual-Link DVI KVMP Switch with Audio

- Dual View Dual-Link DVI support - Share 2 DVI monitors with a single keyboard and mouse, along 4 computers
- Display Emulation Technology reads and remembers the monitor's video parameters so there is no delay when switching between computers
- High resolution - Up to 2560 x 1600
- 2-port USB 2.0 hub for peripheral sharing
- Asynchronous hotkey switching between KVM focus, USB peripheral, and audio ports

Mac | PC Compatible | Sun Compatible | Linux Compatible

Other DVI KVMP Solutions

model: GCS1782/GCS1784

2/4-Port Dual-Link DVI KVMP Switch with 7.1 Audio and Cables

model: GCS1102/GCS1104

2/4-Port DVI KVMP Switch with Audio and Cables

* Product images shown reflects the front and back view for your reference only.

TigerDirect.com CDW PC Connection MacMall altex Computers & Electronics amazon.com newegg.com Fry's Electronics MICRO CENTER

Let's Connect www.iogear.com

TABLE OF CONTENTS

From the Editor

Change in the world of technology is certainly not unheard of. However, the changes we're seeing the Apple Universe have been astounding. What has been going on just this year alone?

- The annual Macworld conference runs for the first time without an official Apple presence. The smaller show proves a success, but begins the challenge to grow anew. (See http://www.macworldexpo.com).

- Apple releases the iPad; it's a huge hit. The iPad outsells just about everyone's expectations. Developers everywhere want to tap into this new market. (See…well, just about any tech-related media).

- A seemingly legitimate next-generation iPhone is discovered left in a bar. Better than a rumor, more detailed specs are uncovered and the speculation begins. (See this month's "Apple's Tough Love" by Michael Swaine).

- Apple announces the details for this year's WWDC. It's iPad focused and entirely devoid of a real Mac or IT track. Even the Design Award for anything Mac OS X related is missing. Speculation ensues (See, this month's "WWDC For Enterprise Admins" by Greg Neagle).

- Jonathan Rentzsch cancels the annual C4 conference. This is primarily due to the lack of the developer community's outrage over section 3.3.1. (See, "C4[3] in Retrospect, MacTech November 2009).

The common thread here—aside from change—is challenge. From Apple themselves down to all of the people in that ecosystem, we're all being challenged in new ways. It's exciting, though, isn't it? It's motivating. It's interesting.

Necessity is the mother of invention, and with that comes opportunity. Apple provides us with both the challenges and opportunities. For example, look at the new capabilities in Mac OS X Snow Leopard that Grand Central Dispatch gives you (see this month's, "Calling All CPUs" by Dave Dribin). What a great opportunity.

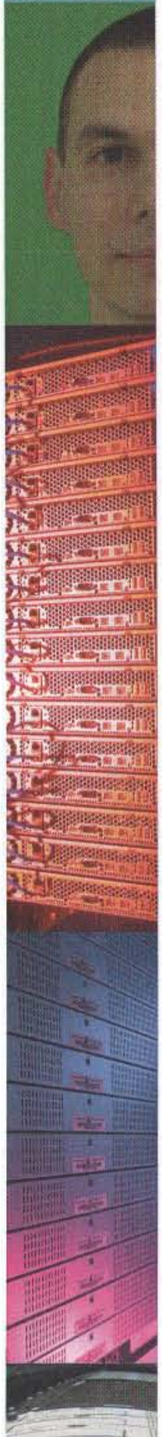
As developers, we should always be questioning and improving our skills. This is a challenge (see "Source Code Metrics on Xcode" by José Cruz).

As System Administrators, we should also be questioning and improving our skills (see, "Dating Files," "Configuring Git" and "Making PKG Files with The Luggage").

No matter our niche, we should be reaching out to and following our peers as a way to foster community and brainstorm (see The MacTech Spotlight featuring Daniel Stødle).

We're all being challenged, motivated and surprised; I wouldn't have it any other way.

Ed Marczak,
Executive Editor



Communicate With Us

Department E-Mails

Orders, Circulation, & Customer Service
cust_service@mactech.com

Press Releases
press_releases@mactech.com

Ad Sales
adsales@mactech.com

Editorial
editorial@mactech.com
(Authors only, no pr)

Accounting
accounting@mactech.com

Marketing
marketing@mactech.com

General
info@mactech.com

Web Site
http://www.mactech.com

In this electronic age, the art of communication has become both easier and more complicated. Is it any surprise that we prefer **e-mail?**

If you have any questions, feel free to call us at 805/494-9797 or fax us at 805/494-9798.

If you would like a subscription or need customer service, feel free to contact MacTech Magazine Customer Service at 877-MACTECH

We love to hear from you! Please feel free to contact us with any suggestions or questions at any time.

Write to letters@mactech.com or editorial@mactech.com as appropriate.

MACTECH®

The Journal of Macintosh Technology

A publication of **XPLAIN** CORPORATION

The Magazine Staff

Publisher & Editor-in-Chief: Neil Ticktin
Executive Editor: Edward R. Marczak
Business Editor: Andrea Sniderman
Ad Director: Bart Allan
Production: David Allen
News: Dennis Sellers
Podcast Producer: Josh Long
Staff Writer: Chris Tangora
Staff Writer: Frank Petrie
drupalmaster: Erik Peterson

Xplain Corporation Senior Staff

Chief Executive Officer: Neil Ticktin
President: Andrea J. Sniderman
Accounting: Marcie Moriarty
Customer Relations: Susan Pomrantz

Columnists

Mac In The Shell: by Ed Marczak
The Road to Code: by Dave Dribin
Swaine Manor: by Michael Swaine
KoolTools/Geek Guides: by Dennis Sellers
MacEnterprise: by Philip Rinehart and Greg Neagle

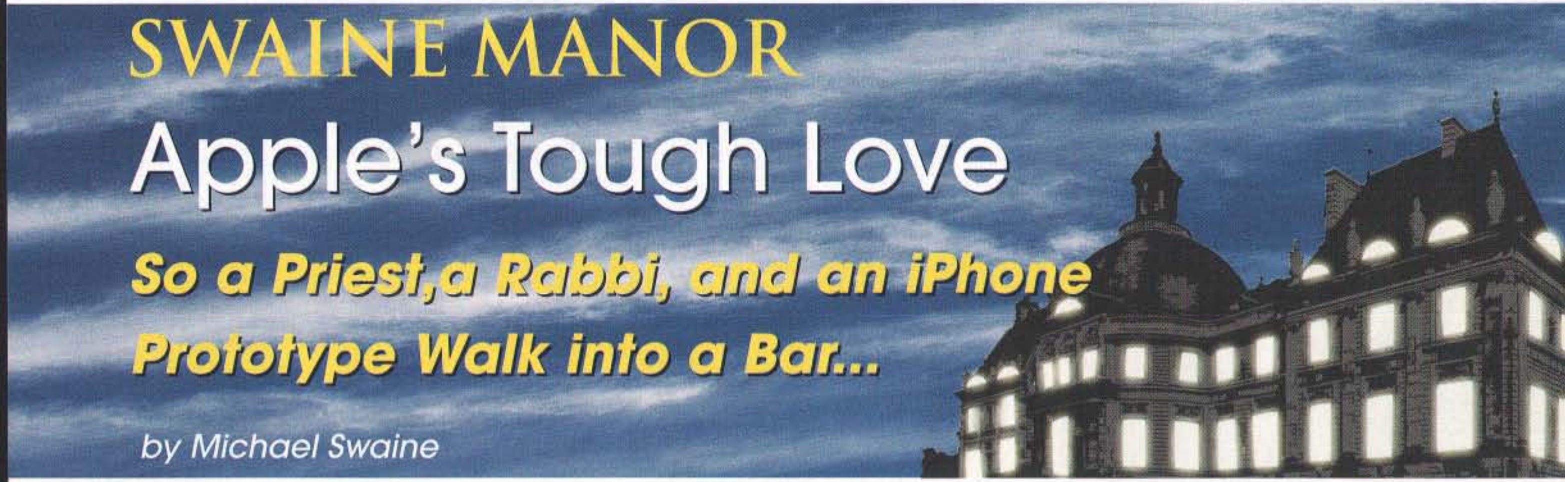
Regular Contributors

José R.C. Cruz, Michael Göbel, Michele Hjörleifsson, Mihalis Tsoukalos
Oliver Pospisil, Rich Morin, William Smith

Canada Post: Publications Mail Agreement #41513541
Canada Returns to be sent to: Bleuchip International, P.O. Box 25542, London, ON N6C 6B2

MacTech Magazine (ISSN: 1067-8360 / USPS: 010-227) is published monthly by Xplain Corporation, 705 Lakefield Road, Suite I, Westlake Village, CA 91361. Voice: 805/494-9797, FAX: 805/494-9798. Domestic subscription rates are $47.00 per year. Canadian subscriptions are $59.00 per year. All other international subscriptions are $97.00 per year. Please remit in U.S. funds only. Periodical postage is paid at Thousand Oaks, CA and at additional mailing office.

POSTMASTER: Send address changes to **MacTech Magazine**, P.O. Box 5200, Westlake Village, CA 91359-5200.



Apple's Tough Love

So a Priest, a Rabbi, and an iPhone Prototype Walk into a Bar...

by Michael Swaine

Tough Love

I'm not one of those nervous nellies who go all frizzy whenever Apple has to slap some blogger's wrist. You know, people like Steve Wozniak (http://news.cnet.com/8301-13579_3-20003384-37.html).

These people just don't understand Apple.

See, Apple loves us. That's why it keeps giving us these magical gifts: the iPad, the iPhone, the iPod, the iPod Sock. Not to mention the eagerly anticipated next version of OS X, code-named Lolcat. No, seriously, we are not to mention it. We're all under universal nondisclosure. Haven't you been reading those license agreements you've been clicking on every time you let Apple install a software update?

Apple *loves* us. Apple is *sad* it had to be so stern.

I'm just kidding. And if you truly believed in the deep and abiding love Apple has for you, you would have known that. But sometimes Apple has to practice tough love.

So when an engineer left a secret prototype of the next iPhone in a Redwood City bar and a college student found it and sold it to a blogger and the blogger went public with the story, Apple naturally got a little miffed, and naturally called in the California Rapid Enforcement Allied Computer Team and the Redwood City Police.

And this led to some unpleasantness (http://abcnews.go.com/print?id=10523213) about jackbooted Gestapos kicking down a journalist's door and seizing his computer and so on. Not what Apple wanted at all. Because Apple *loves* us. Apple is *sad* that it had to be so stern.

Journalism and Dancing

By the way, this story once again raised this whole "is a blogger a journalist?" thing. Let's deal with that.

A lot of bloggers want to be journalists. They think it's a glamorous, noble profession. They think that by being recognized as true journalists they are on their way to a lucrative career. That's right: they think it's a *career*. They clearly haven't been paying attention to what's been going on in journalism the past few years.

There is a legend that once journalists were brave and noble and stood up to corporations and the government. I'm not saying that was never true. I wouldn't know; as a child of television I can't remember anything that happened over three weeks ago. But clearly this legend has nothing to do with journalism today.

Bloggers, take note. You have more integrity and credibility right now than the average journalist, and if you add one more banner ad to your blog, you'll be making more money, too. Why would you want to be a journalist?

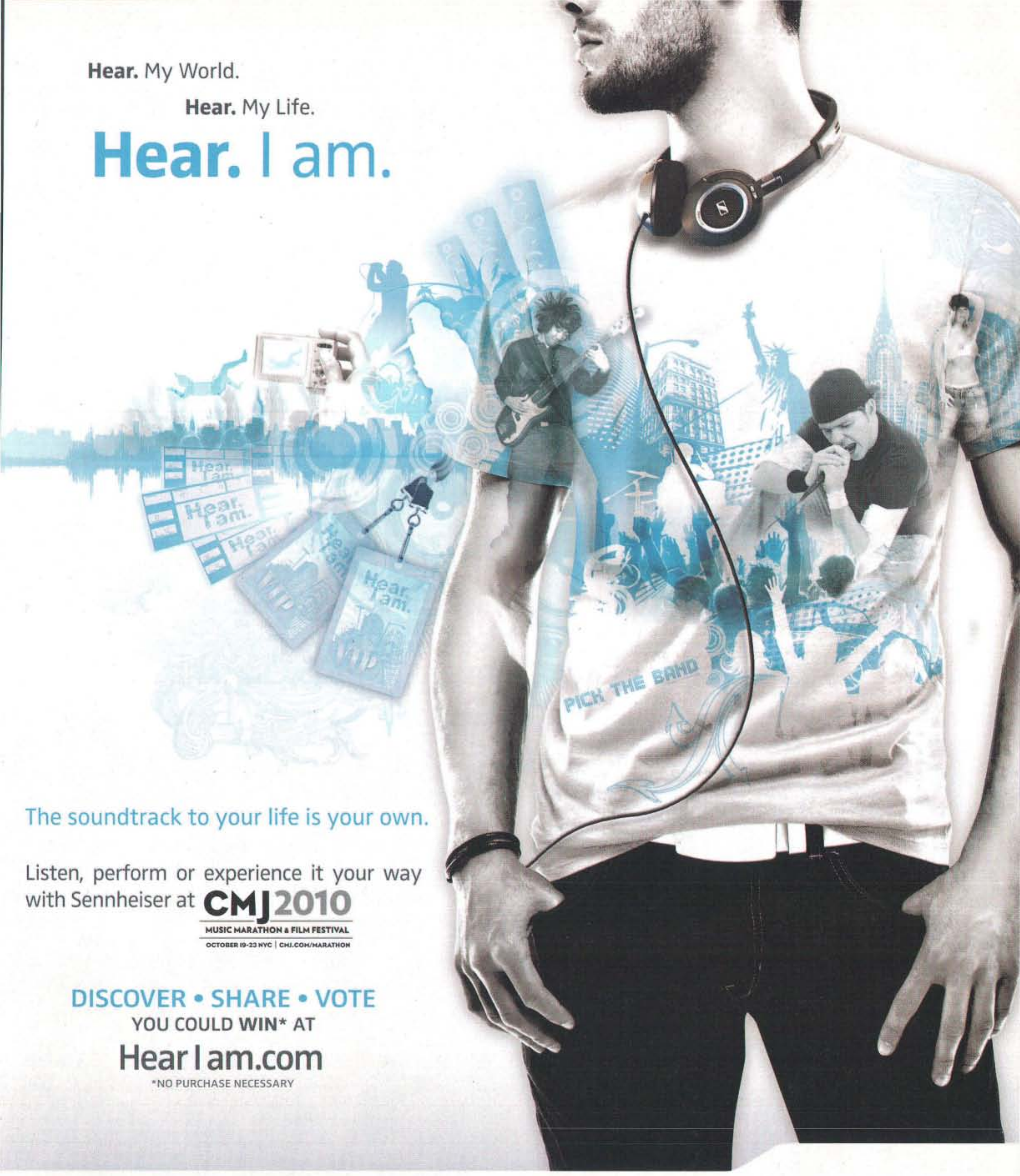
And then there's the Ellen thing (http://news.cnet.com/8301-13579_3-20004186-37.html).

Let's face it: Ellen Degeneris has been pushing it for years, doing her "dancing with the stars" thing without paying royalties to Dancing with the Stars. Now she really went over the edge, making a joke iPhone commercial that suggests that the iPhone user interface might be confusing or hard to use. Excuse me? That's clearly actionable, not to mention a possible human rights crime against potential Apple customers who might be confused by this joke

Hear. My World.

Hear. My Life.

Hear. I am.



The soundtrack to your life is your own.

Listen, perform or experience it your way with Sennheiser at CMJ 2010

MUSIC MARATHON & FILM FESTIVAL

OCTOBER 19-23 NYC | CMJ.COM/MARATHON

DISCOVER • SHARE • VOTE

YOU COULD **WIN*** AT

Hear I am.com

*NO PURCHASE NECESSARY

Headphones
& Microphones

SENNHEISER

commercial and as a result be denied the joy of iPhone ownership. I love Ellen, but she really messed up here.

It's a good thing that Apple called her on this, for her own good as well as ours. Otherwise you *know* where she'd have gone next. Yep, she'd have violated Apple's ownership of dancing silhouettes by fiddling with the lighting when she was doing one of those embarassing-the-guest-by-making-them-dance things she does.

Bullet dodged. Thank you Apple, I'm sure Ellen is saying.

Eternally Vigilant

Now I understand that the publishers of *iPhone App Development: The Missing Manual* are nervous since Apple slapped them with a restraining order until they can determine who took this manual. Not to worry. Redwood City police are on the case, employing the current profile of the Conduit of Stolen Apple Merchandise, derived from The Case of the Phone Left in a Bar. So they're looking for another college student working part time at a church-run community center giving swimming lessons to children after volunteering at a Chinese orphanage.

Or whoever Apple tells them to look for.

An arrest is expected imminently.

Can I just say that I feel a lot more secure now about leaving things in bars? Especially anything with Apple's logo on it. I know Apple's got my back.

I'm sure that in the future Apple will make better use of the GPS and other tracking technology in its devices, so that problems like the missing iPhone prototype can't happen at all. Because they will know exactly where every one of their products is at all times.

So I hope I've made it clear how Apple is looking out for our welfare. Oh, and if all that weren't enough, Apple has provided a business model for the creator of the website founditinabar.com.

Thank you, Apple. We feel the love.

MT

About The Author

Michael Swaine is the former editor-in-chief of* Dr. Dobb's Journal *(http//www.ddj.com) and current editor of* PragPub *(http//www.pragprog.com/magazine), the electronic magazine for pragmatic programmers. You can reach him at mike@swaine.com.

Mac User, Stand Up and Be Recognized

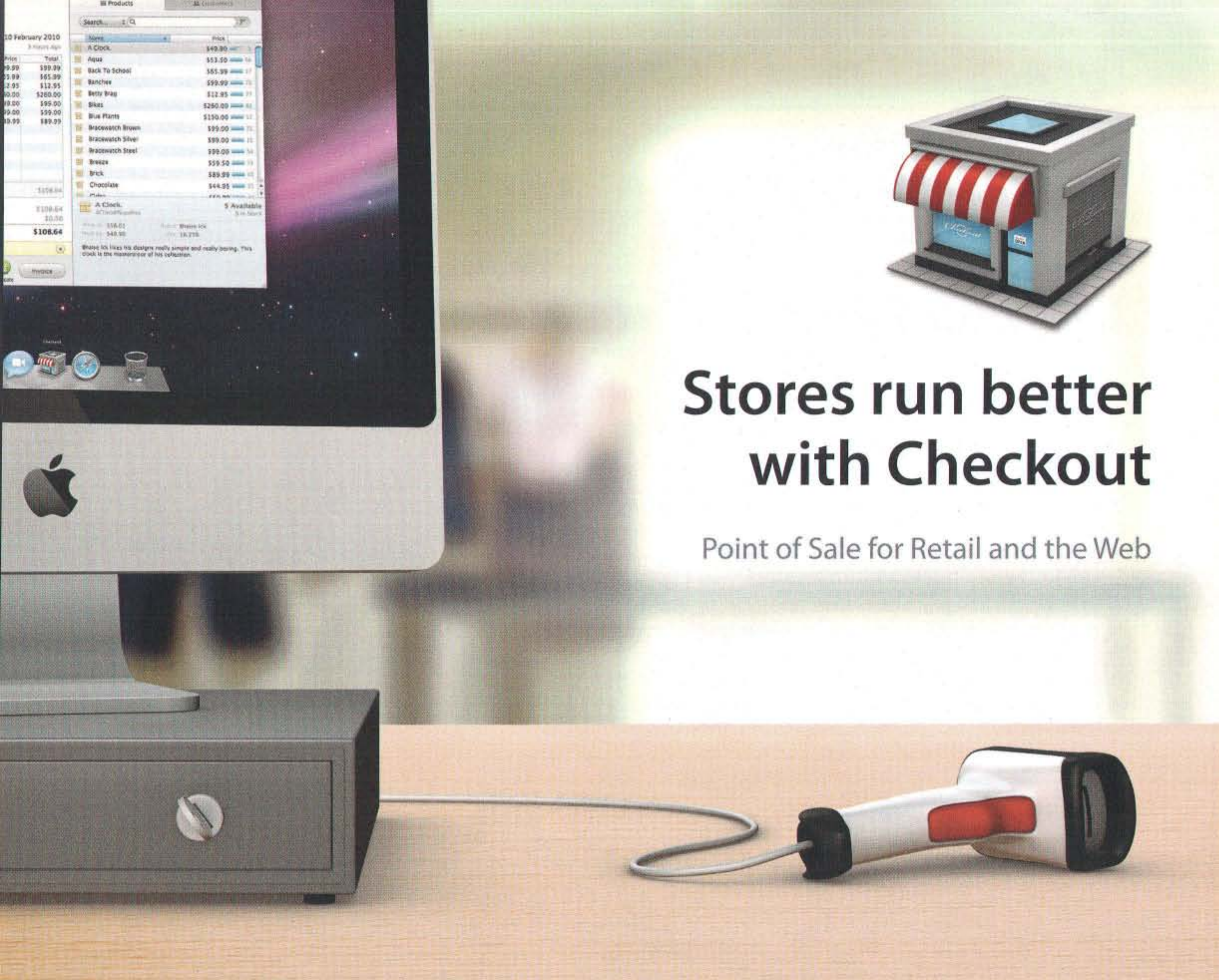
From photo ID's to access control badges, Evolis identifies your employees, students, or visitors on demand.

Fully compatible with Mac OS®, the Evolis range of card printers will always bring you the recognition that you deserve.

TransTech is an Official Evolis Solutions Provider
T: 1.888.843.3643 F: 503.682.0166
www.ttsys.com • Email: sales@ttsys.com
Identifying Solutions

TRANSTECH SYSTEMS

evolis OFFICIAL PARTNER



Name	Price
A Clock.	$49.90
Aqua	$53.50
Back To School	$65.99
Banchee	$99.99
Betty Brag	$12.95
Bikes	$260.00
Blue Plants	$150.00
Bracewatch Brown	$99.00
Bracewatch Silver	$99.00
Bracewatch Steel	$99.00
Breeze	$59.50
Brick	$89.99
Chocolate	$44.95

Price	Total
[illegible]	$99.99
[illegible]	$65.99
[illegible]	$12.95
[illegible]	$260.00
[illegible]	$99.00
[illegible]	$99.00
[illegible]	$89.99

$108.64

A Clock. 5 Available

Bhaise Ick likes his designs really simple and really boring. This clock is the masterpiece of his collection.

Stores run better with Checkout

Point of Sale for Retail and the Web

We all know everything just works better on a Mac®. Now point of sale does too. Do your favorite retailer a favor, tell them to try Checkout for free today.

Starting from **$399**, Complete **hardware bundles** available*

Visit **www.checkoutapp.com** or call **877 788 1202** toll free.

by Edward Marczak

Dating Files

Or, dating files through scripting

Introduction

Correct file metadata is important. Of course, it's sometimes more important than others, personally or otherwise. This month, I'm going to use a problem that I ran into and a basic solution for it that illustrates many bash shell constructs and tools. The problem I faced: getting back the proper time stamp on my digital pictures after a little processing. Read on and see how this was solved and learn some bash scripting on the way.

The Problem

The "problem" that I ran into is that I often shoot bracketed pictures: three pictures for a single scene. One picture is under exposed, one is the right exposure and one is over exposed. These pictures are later combined into a single picture called an "Extended Range" (EXR) or "High Definition Range" (HDR) picture. This allows for much greater control over the depth of ranges you can see within a scene. Adobe Photoshop has built-in support for the .exr file format (although iPhoto and Adobe Photoshop Elements do not). My personal workflow goes like this:

Shoot pictures, both bracketed and standard.
Download pictures from camera.
Sort pictures—bracketed pictures go into their own folder.
Combine bracketed pictures into .exr file.
Photoshop cleanup.
Save as PNG file.

"So what's the problem," you ask? Step 4 takes the original 3 (or more) JPG pictures from the camera and creates a new .exr file from them. This new picture file has no concept of the metadata that accompanied the original JPG pictures from the camera. It only strays further from there.

How I'm handling this: use the create date from one of the original JPG files and applying it to the workflow's resultant PNG file. Even if you don't have this *specific* problem, we're looking at a pretty generic solution here. The formal flow chart in Figure 1 may explain it better:

Most admins have seen this pattern when wanting to drop a file in every user's home directory—or to clean up files in a directory structure. Starting at the root of a volume, one could process an entire disk. This may be for clean up, or even for search. Cataloging all audio files on a given volume? Just process the whole thing one directory at a time.

Let's get to the script and commentary.

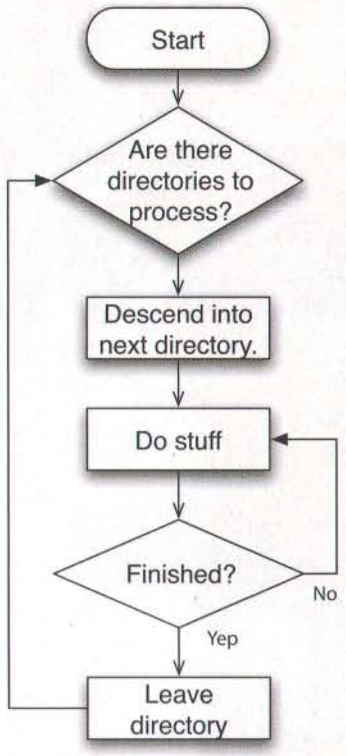

Start
Are there directories to process?
Descend into next directory.
Do stuff
Finished?
No
Yep
Leave directory

Figure 1 – A common, generic workflow

The Solution

Before venturing out, some disclaimers: First, I promised to write scripts for this column in bash for a few months. In Real Life™, I'd actually write this in Python. Secondly, the script, as presented, is a bit hacky. I'll mention ways to improve it in the running commentary. Finally, the workflow that I use for taking and processing pictures is less than ideal. I know there are some better methods and products to use, but at the moment, this workflow suits me just fine (and allowed me to write this column).

A basic version of the script I've been referring to is shown in Listing 1.

Listing 1: photo_fix.sh

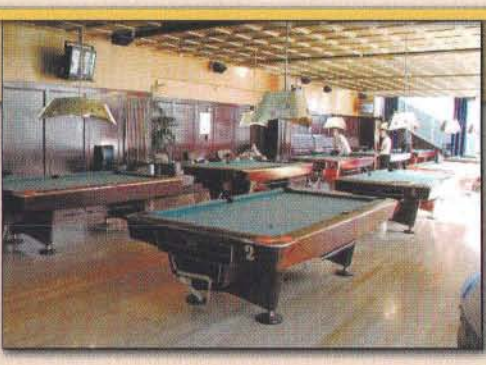
```
#!/bin/bash

IFS="
"

listdir() {
  for i in $(ls -1 "${1}"); do
    echo "Processing ${1}/${i}"
    if [ -d "${1}/${i}" ]; then
      cd "${1}/${i}"
      # Locate the png file - expecting only 1 per directory.
      png_file=$(ls -1 *.png | head -1)
      if [[ ${png_file} == "" ]]; then
        # If there is no png file, the rest is moot - bail
        echo "No png file in this directory - skipping"
        cd -
        continue
```

JILLIAN'S AT METREON

The Perfect Place to Host your Next Event!

The Main Dining Room

- 500 Guests Reception Style
- 175 Guests for Plated Lunch or Dinner
- 300 Guests Buffet Style
- A Great Spot for Off-Site Company Meetings
- 50-foot Video Wall and Bar
- State of the Art A/V Hook-ups



The Billiard Lounge

Featuring 10 Brunswick Gold Crown Tournament Style Billiards Tables and Can Accommodate:

- 150 Guests Reception Style
- 120 Guests Plated Meal
- Can Be Sectioned Off for Smaller Gatherings With Use of the Outdoor Patio

The Martini Bar

- Up to 50 Guests Reception Style
- 35 Guests Buffet Style
- Up to 40 Guests for Sit-Down Events
- Complete With a Plasma Screen TV
- Wireless Internet Connection

The Lower Dining Room Can Accommodate:

Up to 80 Guests for Sit-Down Events

100 Guests for Receptions – 75 Guests for Theater-Style Events

Contact our Event Sales Manager:
415.369.6101/ ecsanfrancisco@jilliansbilliards.com

```
      fi
      # Info for source JPG
      info_file=$(ls -1 *.JPG | head -1)
      info_fileinfo=$(stat -s "${info_file}" | awk '{print
$12}' | sed 's/st_birthtime=//')
      touchdate=$(date -j -r ${info_fileinfo}
"+%Y%m%d%H%M.%S")
      echo "Current file here is ${info_file} with date
${touchdate}"
      echo "Touching ${png_file}"
      touch -t ${touchdate} "${png_file}"
      cp -pv "${png_file}" "${FINAL}"
      cd -
    fi
  done
}

main() {
  if [[ ${1} == "" ]]; then
    echo "You must supply a root directory to process."
    exit 1
  fi
  if [[ ! -d ${1} ]]; then
    echo "${1} is not a directory"
    exit 2
  fi
  BASEDIR="${1}"
  FINALDIR="${BASEDIR}/Final/"
  listdir "${BASEDIR}"
}

main "${@}"
```

Let's dissect it and see what's going on.

The Source

The script starts with the standard shebang line pointing to a bash interpreter. This, of course should match your environment. If you have an alternate version of bash, you can specify it here, but this script certainly doesn't need it.

The very first line set the IFS variable:

```
IFS="
"
```

IFS stands for "Internal Field Separator." This is a list of characters that bash uses to know when to tokenize a string. Essentially, what separates fields, or "word boundaries," in a mass of text? By default, IFS is set to whitespace (space, tab and newline). This works nicely for most cases, but, as you see, it can be changed. Why do we change it? Simple: directories with spaces in the names. Leaving it set at the default will cause the parser to recognize the space in a single filename as a new field. The directory listing we're going to produce will have a newline character to separate each field.

The next line starts a bash function. In this case, it's called listdir. We'll come back later to this line and the lines it comprises, which are all lines between the curly-braces (24 lines in total). The same goes for the main() function which follows. The bash interpreter doesn't execute the lines in a bash function until called. In this case, the interpreter executes the IFS line, pointed out earlier, then sees a function definition

SmallTree

The Mac Networking Experts

Manage your workflow better with Shared Storage that everyone can afford

The Mac Networking Experts
1.866.782.4622
Home
Downloads
Contact Us
Simple, Affordable, Effective shared storage solutions for everyone
GraniteSTOR
ST-RAID
Final Cut Pro & Video Solutions
abcSAN
ProTools & Audio Solutions
What is ST-RAID
What is abcSAN

Check out all our Ethernet Shared Storage Solutions at

GraniteSTOR.com

Winner of the 2010 Vidy Award

www.small-tree.com • 1.866.782.4622 • 866.STC.4MAC
7300 Hudson Blvd., Suite 165, Oakdale, MN 55128
www.GraniteSTOR.com

☑ Android
☑ BlackBerry
☑ Palm Pre
☑ Symbian
Sync ☑ Windows Mobile with Mac

Wirelessly

Your smartphone frees you from the land-line phone in your office. Your Bluetooth headset frees you from cabled earphones and mic. Let The Missing Sync free you from tethering your phone to your computer every time you want to sync. Sync over Wi-Fi or Bluetooth.*

Automatically

When your smartphone is near your computer, sync happens without having to think about it. Photos you snap on your phone, changes to contacts, new appointments, new music - they'll sync automatically between your phone and computer. It's just like magic!**

www.markspace.com/SyncIt

mark/space

*Wireless capabilities vary among phones, so The Missing Sync features will vary, as well.

**Proximity Sync feature not yet available for The Missing Sync for Windows Mobile.

bash Functions

Functions are essentially scripts in and of themselves. If they're part of another script, they're scripts-within-scripts. The only difference from any other script is where it is stored. A script that you run as a utility will get stored on disk and be called as needed. A function, though, gets stored in memory. This can happen with functions that are created on demand, or functions defined in a larger script.

Functions are important in scripts as they help organize large scripts into smaller, more manageable chunks.

(listdir), notes its location but doesn't run it. It then sees another function (main), notes that it exists and then skips over executing it. The next line, which is also the last in the script, *is* ready to be executed:

```
main "${@}"
```

This line simply runs the main() function, passing it any arguments that were passed into the program itself. The $@ variable is special to bash. It represents each argument passed into the program or function as a separately quoted word. We take this bundle and pass it along to our main() function. Each of these variables can be accessed by position. $1 is the first argument, $2 is the second and so on. $0 is the name of the script as called from the command line. Calling a function means transferring execution to that function, which means we jump to the main() function from here.

In this script, the main function just performs some extremely basic sanity checks and then let's the real work begin. What type of sanity checks? To start with:

```
if [[ ${1} == "" ]]; then
  echo "You must supply a root directory to process."
  exit 1
fi
```

We make sure that at least *something* is passed in as an argument for which directory to process. We're only ever expecting one argument here, so this meets our needs. We could go off and check for more and warn if there are any trailing arguments, if we needed to be really stringent. From there, we run this check:

```
if [[ ! -d "${1}" ]]; then
  echo "${1} is not a directory"
  exit 2
fi
```

Here we use the "-d" test to verify that what was given to us actually *is* a directory. The exclamation point negates the test, so, this actually reads, "if the variable $1 is *not* a directory, then..." Notice that both of these conditions exit with a specific number—the *exit code*. It greatly helps to have an exit code to know why and where things went wrong.

If both of those checks pass, we set two variables, $BASEDIR and $FINALDIR and then start the real work by calling the listdir() function. We pass along $BASEDIR as an argument.

The Heart of the Solution

All of the important work in this script is done in **listdir()**. It begins with a for loop that encapsulates the entire function:

```
for i in $(ls -1 "${1}")
```

This is a standard **for** loop that iterates over the list produced by the command supplied; in this case, the command is the listing (**ls**) of the first argument passed into the function (**$1**). If you were to run the ls command as supplied, it would look something like this:

```
$ ls -1 /Volumes/homes/SP2010-Master/zHDR
Alps 1
Alps 3
Alps 4
Chinese Gardens-Pagoda
Kapellbrücke-Bridge View
Lucerne-Gutsch Hotel
Rigi Overlook Lucerne
Scooters
Uto Kulm View
Zürisee Boathouse
```

Each name in the list—spaces and all—is on its own line, separated from the next by a newline character. Again, this is

Command Substitution

Command substitution is one of the most important concepts to anyone that hopes to write really useful shell scripts. The syntax is simple:

```
$(command)
```

The command(s) inside of the parenthesis are run and standard output is returned. The example given in Listing 1 captures the output from the **date** command. A simpler example could capture the default output of the **date** command, yielding the current date:

```
date_now=$(date)
```

After this is run, the variable $date_now will contain the output from **date**. Of course, you can use any command available to you. You can pipe commands and even nest command substitution. Use this incredibly flexible aspect of the shell to your advantage wherever possible.

imagine. amaze. inspire.

you think it. you create it.

REAL Studio is the powerful, easy-to-use tool for creating your own software. At REAL Software, we call it a problem solver. You've probably said, "Wouldn't it be great if there was an application that ..." REAL Studio fills that blank.

www.realsoftware.com

Download your free 30-day trial edition today! Or buy now - REAL Studio comes with a 90-day money back guarantee.

REAL Studio compiles native applications for Mac OS X, Windows and Linux from one set of source code. Each version of your software looks and works just as it should in each environment. You can even create a Universal Binary with one mouse-click.

REAL Studio

why we set $IFS to a newline (and **only** a newline). At the beginning of each iteration of the `for` loop, `$i` will contain the next name in the list.

The echo command is just there for our own information and sanity, as it prints which name in the list it's looking at. Each name in the list is hopefully a directory. However, as this is a manual process, we may have some stray files in our master directory. Only if the entry is a directory do we bother trying to do any processing. If it is a directory, we change the current working directory into it (the `cd` command). From there, we start some processing specific to our workflow.

First, we look for a PNG file. Since this is the file we're meant to update, if it doesn't exit, there's no processing to do. So, we look for a PNG file in the current directory by capturing the output of the subcommand.

If we do *not* find a PNG file and decide to bail out, there are two things we need to do: restore our former position in the directory tree and forget the reset of the loop. For the first issue, we can pass the `cd` command a single hyphen ("-") as an argument to return to the previous directory. We could be more stringent about this by capturing the output of the `pwd` command prior to changing and then explicitly changing back to that directory. However, since we're never more than one level away from anything we want to process, this method works just fine. To escape from the loop *but continue to process the loop*, we use the `continue` command. `continue` simply stops the loop and jumps back to the top, processing the next item in the list. (This is different than the `break` command, which jumps out of a loop and stops processing that loop altogether).

Now that we are sure we have work to do, here's where we do it. First, we grab the first JPG file that we find in the current directory:

```
info_file=$(ls -1 *.JPG | head -1)
```

While there will be multiple JPG files, they're all close enough in time to choose any one of them as a source for the time stamp. Next, we go get that time stamp. Using the `stat` command, we can find, among other things, the creation date of the JPG file in a format that's useful to us without having to do a lot of parsing. This particular line in the script can be a little confusing so, let's look at it a little more closely:

```
info_fileinfo=$(stat -s "${info_file}" | awk '{print $12}' |
sed 's/st_birthtime=//')
```

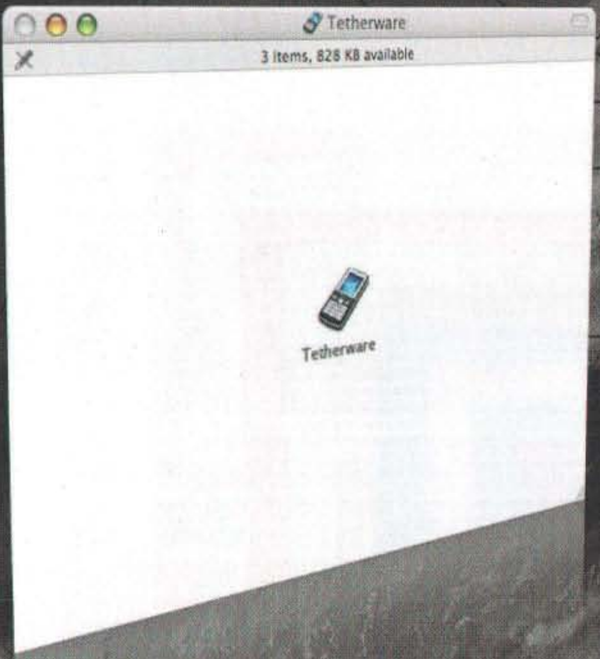
We're capturing the output of a string of commands in the $info_fileinfo variable. The subcommand runs three commands that are piped together. The first command in the chain is stat:

```
stat -s "${info_file}
```

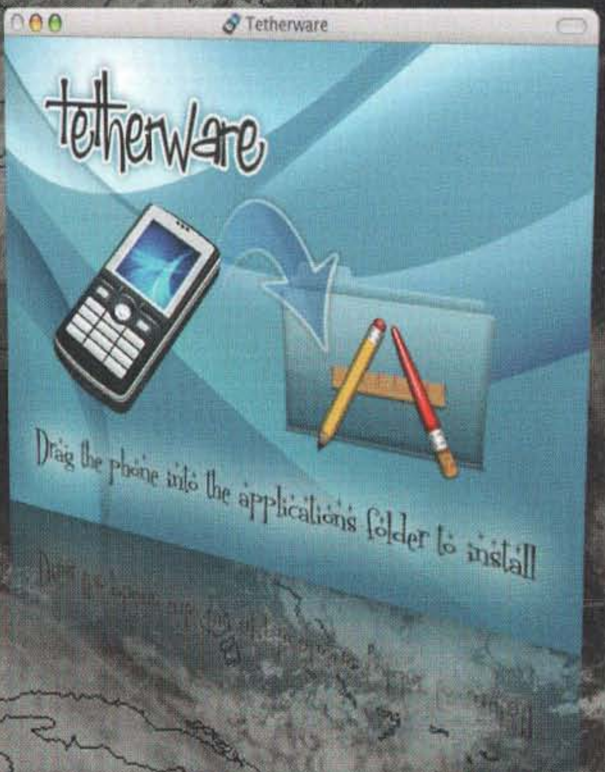
yields:

Is great looking web design an alien concept?
Freeway writes flawless and standards-compliant code, so humans can create out-of-this-world web sites.
Professional features, intuitive workflow and excellent earth-based support mean your outer limits are bounded only by imagination.
Visit www.softpress.com/alien and be abducted by Freeway for 30 days.
You'll soon see why Freeway is the indispensable web-design tool for all carbon-based life forms.
freeway pro
freeway express
J. F. Winters
Macworld
Mac Observer
Mac Life

Does your DMG look damaged?



DMG built WITHOUT FileStorm in OS X 10.6 displayed on 10.5 or earlier



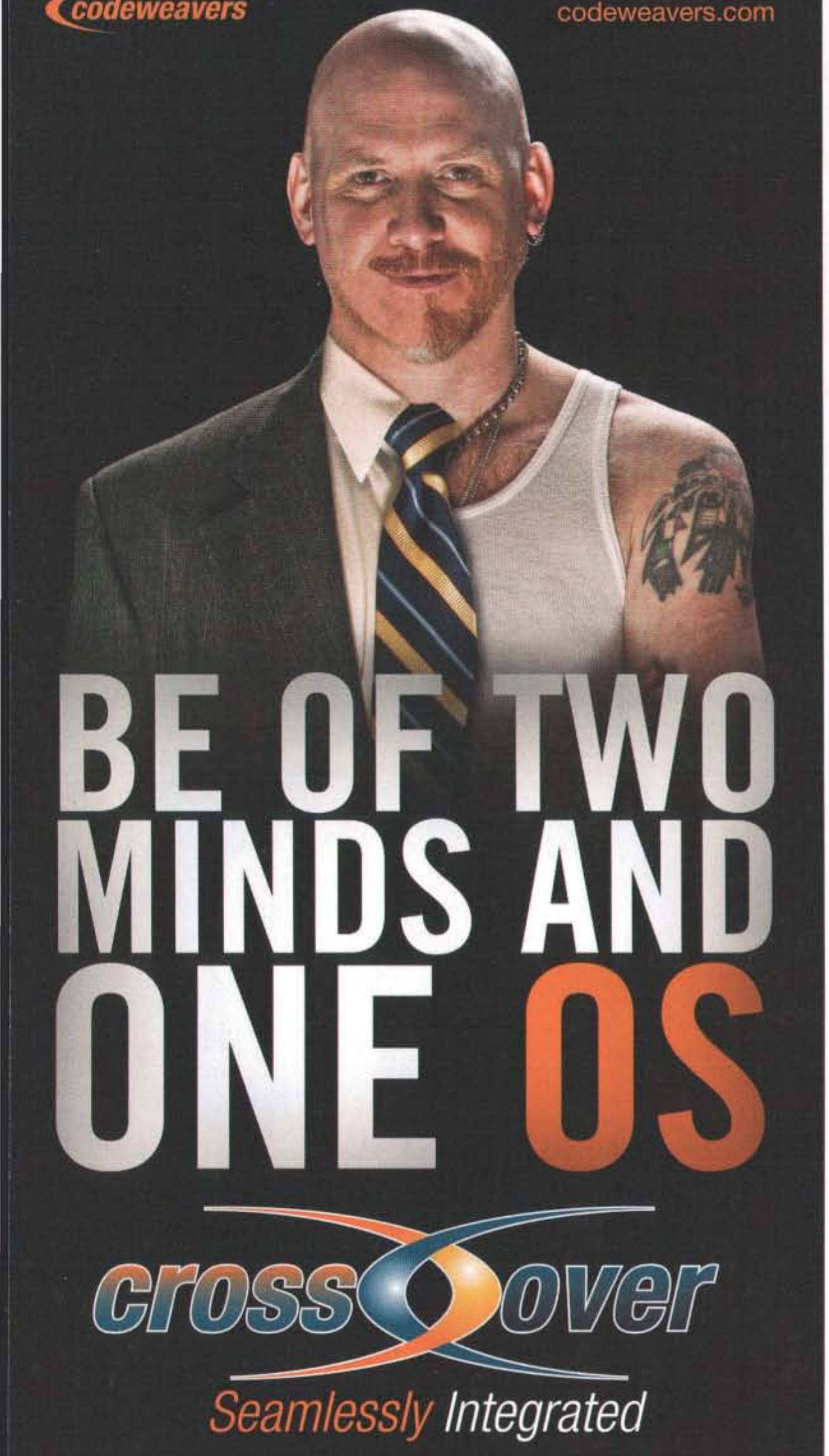
DMG built WITH FileStorm in OS X 10.6 displayed on 10.5 or earlier

FileStorm from MindVision Software.
For when your image is hurting your image.

Purchase and download now
www.mindvision.com/mactech

codeweavers

codeweavers.com



crossover

Seamlessly Integrated

NEVER RUN WINDOWS AGAIN.

- Run Windows applications on your Intel Mac without a copy of Windows!
- No RAM/CPU/Performance hit, unlike other emulators
- Ideal for both practical applications and gaming
- No need to reboot

WINDOWS SOFTWARE DEVELOPER? LET US HELP YOU PORT YOUR APPLICATION TO THE MAC!

GET 20% OFF INSTANTLY

"Wanna help me prove to my coworkers this darn ad was worth it? Next time you order from **codeweavers.com**, include the dealcode **MacTechWWDC,** and we'll take 20% off the price of your CrossOver purchase. Tell 'em Jon sent you!"

```
st_dev=234881028 st_ino=2124429 st_mode=0100640 st_nlink=1
st_uid=79621 st_gid=80 st_rdev=0 st_size=1089060
st_atime=1273100647 st_mtime=1260487255 st_ctime=1260487255
st_birthtime=1260487128 st_blksize=4096 st_blocks=2128
st_flags=0
```

The "-s" switch to stat gives us this 'shell friendly' output, making it easier to parse. There's a lot of information there, from the file's inode and mode to the create and modify dates. We're most interested in the file's "birthdate," which is when the file inode was created. So, this output is piped into awk:

```
awk '{print $12}'
```

which gives us back just the 12th field (awk does not use the shell $IFS variable to determine field boundaries). We then use sed to strip out the label and just grab the value:

```
sed 's/st_birthtime=//'
```

However, our $info_fileinfo variable now contains the time as a Unix timestamp. We need to convert that into a format that **touch** can use. The **date** command can perform this conversion for us:

```
touchdate=$(date -j -r ${info_fileinfo} "+%Y%m%d%H%M.%S")
```

The **-j** switch instructs **date** to not try to set the system date. The **-r** switch takes the Unix-style timestamp (seconds since the Unix Epoch of 1970) and prints the result. The result is printed in any format we choose. **date** supports the same notation as strftime *and* arbitrary text. See the man page for strftime for a full list of format substitutions. The ones we've used in the example script are:

%Y – print the year, including the century (e.g. 2010).
%m – print the month as a decimal number with leading zero
%d – print the day of the month as a decimal number with leading zero
%H – print the hour in 24 hour time format (00-23) with leading zero.
%M – print the minute (00-59) with leading zero.
. – That's a literal 'dot'. This text is not substituted.
%S – print the seconds with leading zero.

We capture this touch-compatible date into the variable $touchdate. Just to be clear: only items preceded with '%' are substituted – everything else in the format string is passed through untouched. For example, if we were updating actual EXIF information in a JPG file, the date format used in the "creation" metadata has a different format. It separates each part of the date with a colon, while the time and date portions are separated with a space. We could use the following format if we were updating EXIF metadata:

```
exif_date=$(date -j -r ${info_fileinfo} "+%Y:%m:%d
```

HoudahGeo

Know where you took that photo

Geocode your photos

GPS optional

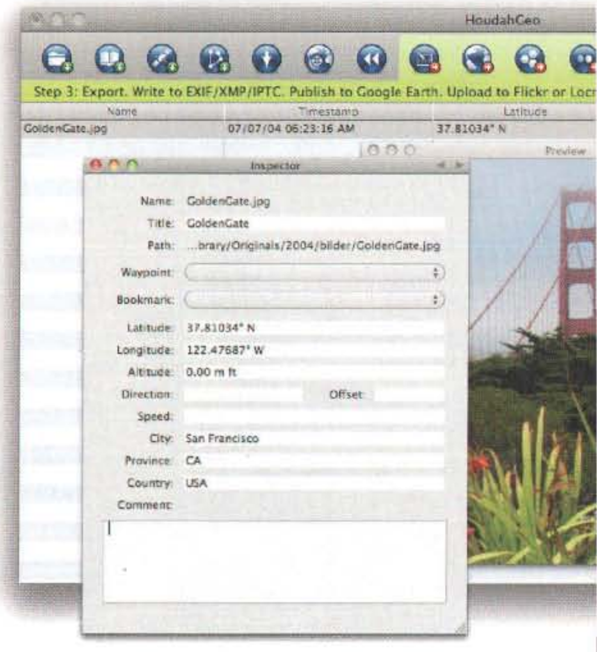
iPhoto '09 Places

Aperture & Lightroom

JPEG, RAW & XMP sidecars

Google Earth. Flickr. Locr

Geocode using iPhone


HoudahGeo

Step 3: Export. Write to EXIF/XMP/IPTC. Publish to Google Earth. Upload to Flickr or Locr

Name	Timestamp	Latitude
GoldenGate.jpg	07/07/04 06:23:16 AM	37.81034° N

Inspector

Name: GoldenGate.jpg
Title: GoldenGate
Path: ...brary/Originals/2004/bilder/GoldenGate.jpg
Waypoint:
Bookmark:
Latitude: 37.81034° N
Longitude: 122.47687° W
Altitude: 0.00 m ft
Direction: Offset
Speed:
City: San Francisco
Province: CA
Country: USA
Comment:

MacBreak Weekly PICK OF THE WEEK Leo Laporte

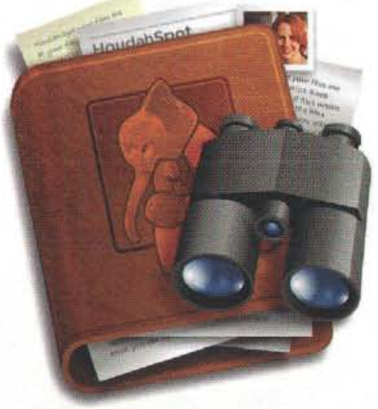
★★★★★ (4.9) MacUpdate

REFERENCE studio.messlinger.com 4/2009 Geotagging

20% OFF

www.HOUDAH.com/store

Use coupon code MACTECH2010 by June 30



HoudahSpot

Find files fast! Spotlight unleashed

Detailed queries to pinpoint files

Include / exclude locations

Query any attribute

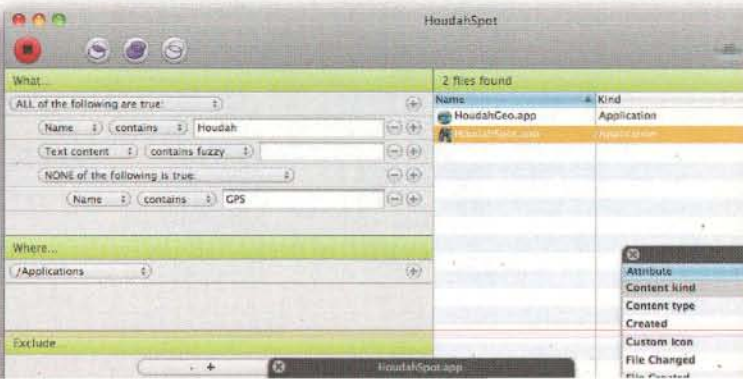
Any attribute as column

Quick access via saved searches

Unlimited customizability

The Spotlight front-end most users don't know they are missing

"...give HoudahSpot a test run—you may find it to be the perfect companion for Spotlight." **Macworld**


HoudahSpot

What...
ALL of the following are true
Name contains Houdah
Text content contains fuzzy
NONE of the following is true
Name contains GPS

Where...
/Applications

Exclude

2 files found

Name	Kind
HoudahGeo.app	Application
HoudahSpot.app	Application

versiontracker EDITORS' PICK

Vous et Votre Mac

MacBreak Weekly PICK OF THE WEEK Merlin Mann

```
%H:%M:%S")
```

Again, notice the parts that are *not* preceded with a percent sign: The colon and space characters are passed through as-is to produce a date that looks like this:

```
2010:06:11 09:01:11
```

This is a wonderful feature of the formatting in the date command.

The next two "echo" lines in the script are merely for informative purposes while the script is running. In your script, you could opt to direct these to a log or not print them at all, depending on your needs. Perhaps you could support a "verbose" flag and only print if that flag is supplied.

The touch command is really doing all of the work we want. Of course, it can only do so once it has all of the prerequisite information:

```
touch -t ${touchdate} "${png_file}"
```

The -t flag uses a timestamp to set the file's date, rather than the current time. From there, we simply copy the file to its final destination:

```
cp -pv "${png_file}" "${FINAL}"
```

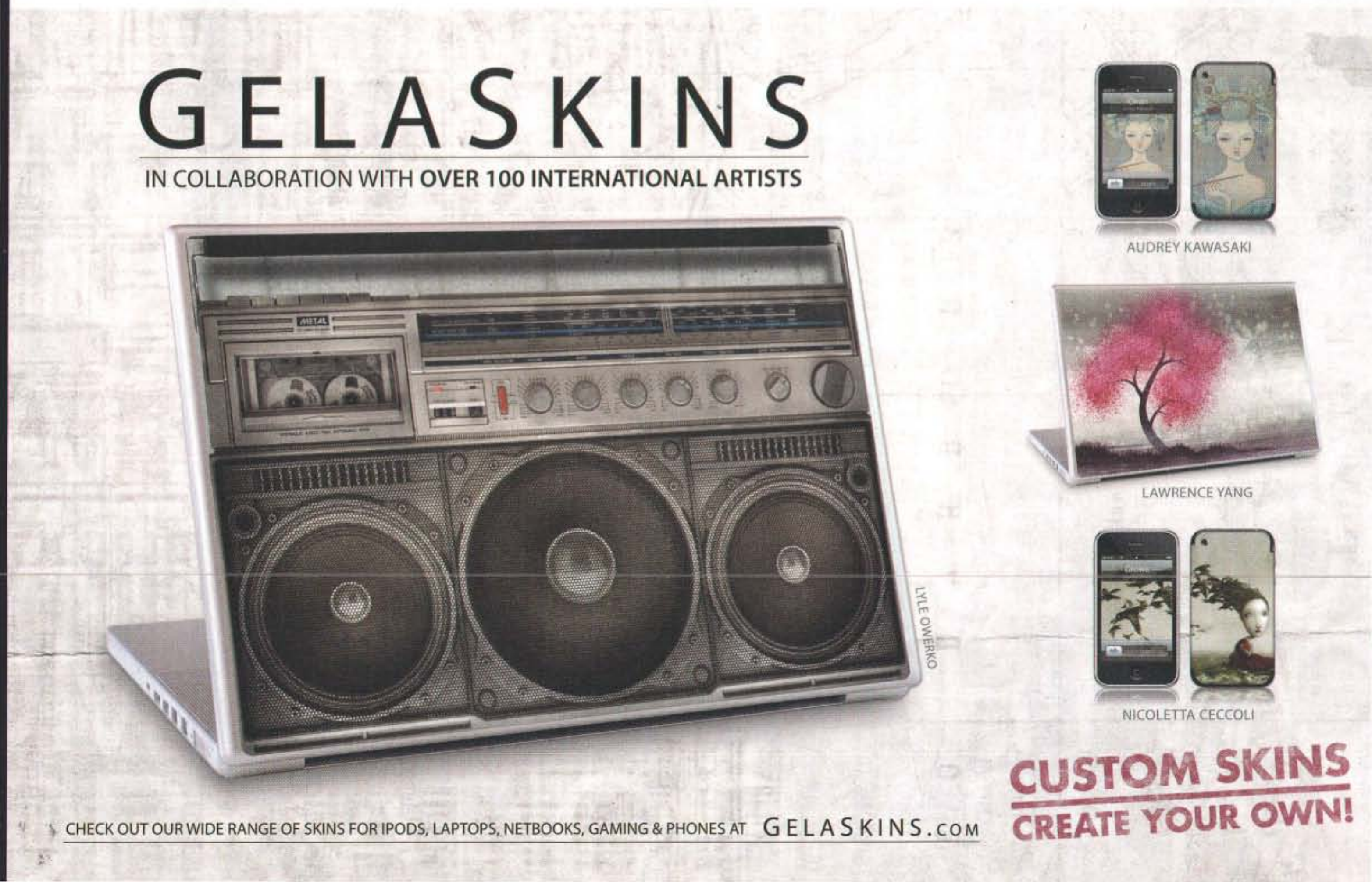
cp's -p flag preserves filesystem metadata (see the man page for details), and the -v switch prints files as they are copied.

From there, we need to go back to the directory we came from; we do that with **cd -**. That ends one iteration of the loop. If there are more directories to process, we start all over again. Otherwise, the script finishes up.

Fire It Up

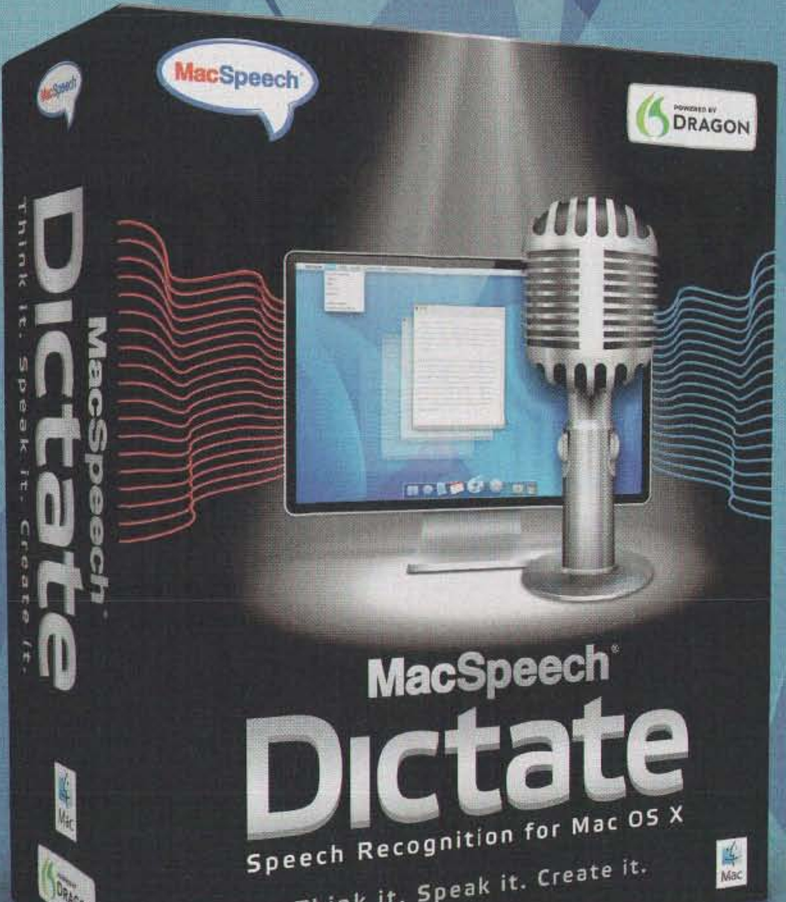
Of course, now you want to run this script yourself, so, here's how. You'll either need to drop this script someplace already in your path (see your $PATH variable), or, you'll need to specify the full path to the file. You'll also then need to pass in the name of the base directory to process—either the absolute or relative path. Here's a sample run that only shows two entries being processed:

```
$ ~/bin/photo_date_fix.sh/photo_date_fix.sh zHDR
Processing zHDR/Alps 1
Current file here is DSCN4723.JPG with date 201004110814.44
Touching Alps 1.png
Alps 1.png -> ../Final/Alps 1.png
/Volumes/homes/SP2010-Master
Processing zHDR/Kapellbrücke 1
Current file here is DSCN4750.JPG with date 201004111059.05
Touching Kapellbrücke.png
Kapellbrücke.png -> ../Final/Kapellbrücke.png
/Volumes/homes/SP2010-Master
```


GELASKINS
IN COLLABORATION WITH OVER 100 INTERNATIONAL ARTISTS
LYLE OWERKO
AUDREY KAWASAKI
LAWRENCE YANG
NICOLETTA CECCOLI
CUSTOM SKINS
CREATE YOUR OWN!
CHECK OUT OUR WIDE RANGE OF SKINS FOR IPODS, LAPTOPS, NETBOOKS, GAMING & PHONES AT GELASKINS.COM

You talk. It types.

MacSpeech Dictate
Premier speech recognition for the Mac.


MacSpeech
POWERED BY DRAGON
MacSpeech Dictate
Speech Recognition for Mac OS X
Think it. Speak it. Create it.
Mac

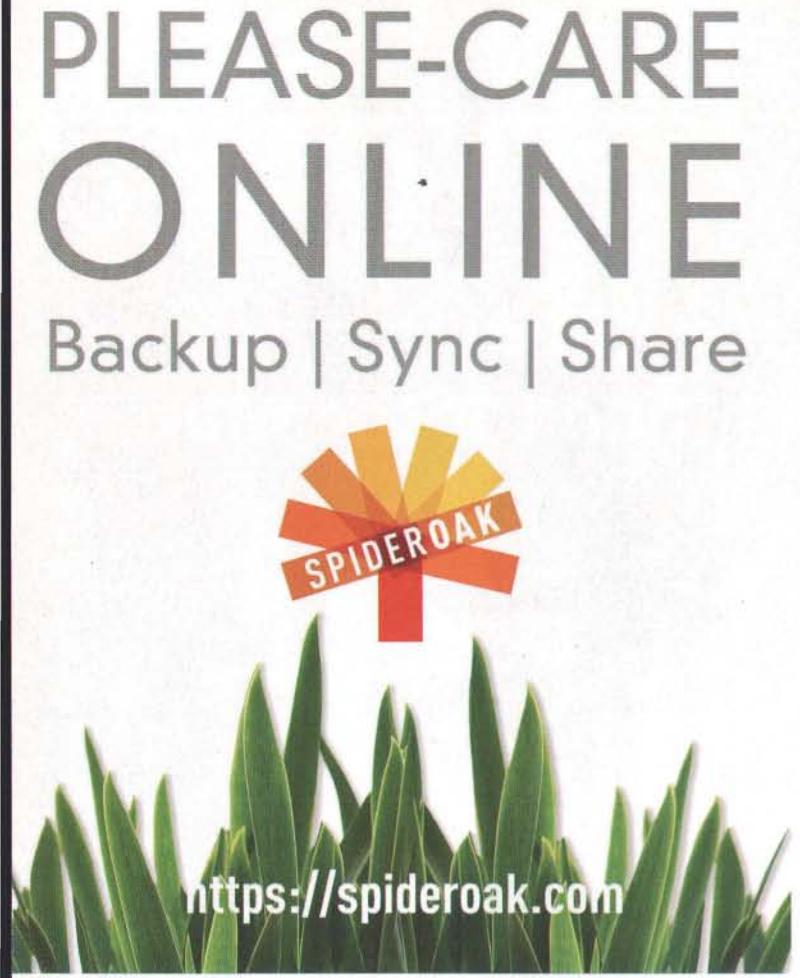
www.macspeech.com

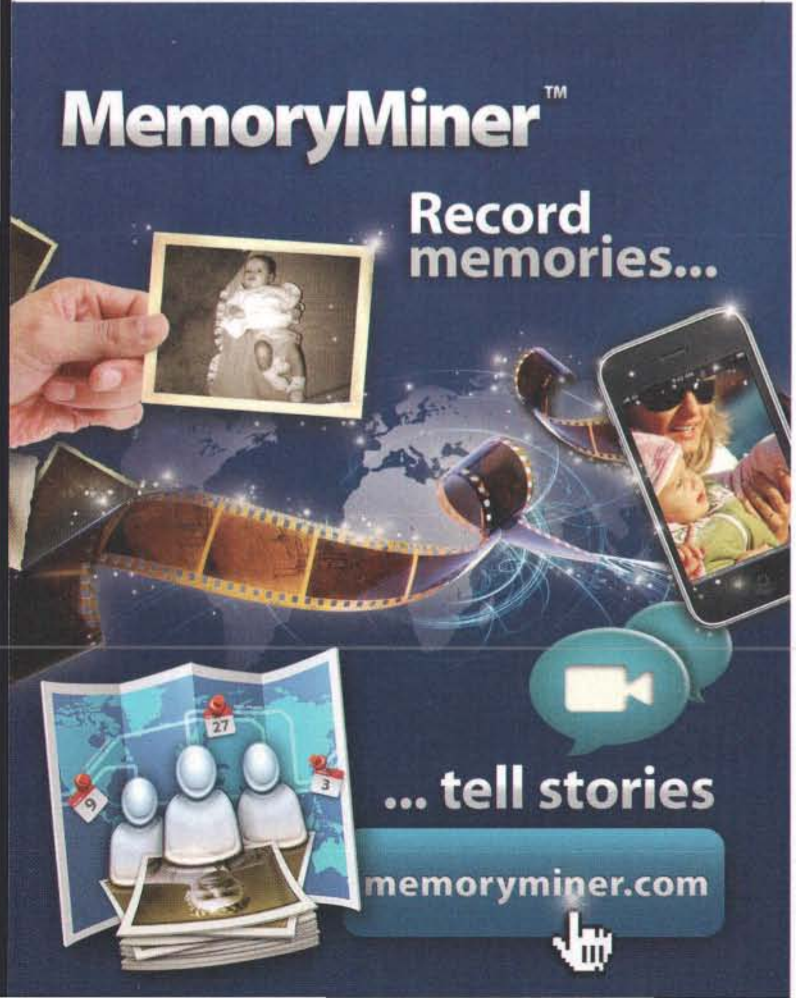
Available from MacSpeech, Apple, and other fine Macintosh retailers.
Visit the MacSpeech website for a complete retailer listing.

MacSpeech®

~~Type~~. Talk!

POWERED BY DRAGON





As mentioned earlier, you may want to clean up or suppress the output of this program.

Improvements

The script, as presented, does everything it needs to do. There are some different ways we could achieve what we need. Some would consider these improvements, so I'll touch on them here.

First, we could use the **find** command to generate the list of directories to process. The **find** command is extremely powerful and designed to do just that: find files on the file system. The general format of find is:

```
find [where] [what] [action]
```

The find command needs a "where" in the form of a path to be directed where to start the search. Using '/' searches the entire file system. The "what" is an expression that defines what to look for. You can search by name, or, purely by metadata information. For example:

```
find . -iname "*apple*" -print
```

...finds any file name in the current directory tree (and in all subdirectories) that contains "apple", case insensitive and then prints the matches to standard out.

As shown, there's also an "action" associated with the results. **find** needs to know what to do with matches it finds. Very often, **-print** will be just what you're looking for, which just prints the results to standard out. Rather than print, **find** can also **-delete** or **-exec**, just to name two other actions. I'll cover find a bit more in future articles.

A second improvement we could make actually impacts the entire workflow. Rather than output to PNG files, we could use JPG files, which support a different set of metadata. We can take advantage of this metadata using **sips**—Apple's "scriptable image processing system" built into Mac OS X. **sips** is a command-line utility that can alter images or their metadata. For example, if you were to take a photo file—typically a JPEG files—directly from your camera and use sips to examine it, you'd find something like this:

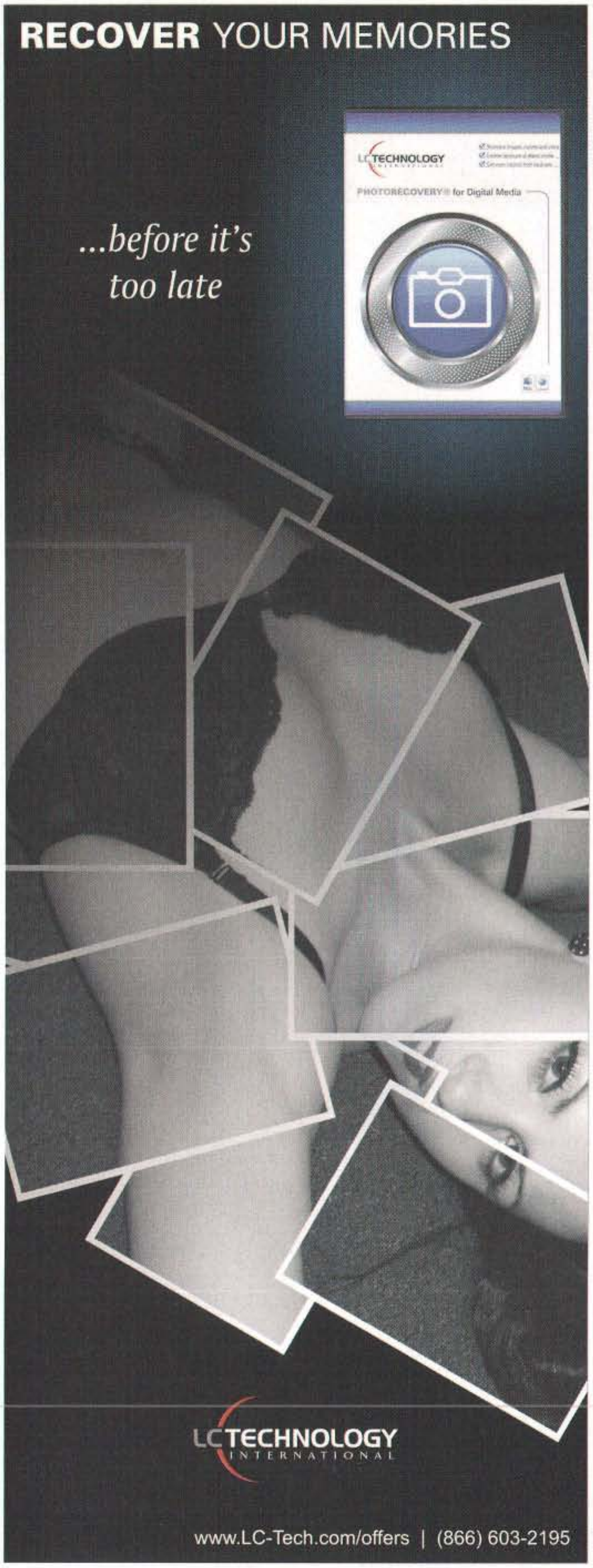
```
$ sips -getProperty all DSCN4551.JPG
/Volumes/homes/SP2010-Master/DSCN4551.JPG
  pixelWidth: 3872
  pixelHeight: 2592
  typeIdentifier: public.jpeg
  format: jpeg
  formatOptions: default
  dpiWidth: 300.000
  dpiHeight: 300.000
  samplesPerPixel: 3
  bitsPerSample: 8
  hasAlpha: no
  space: RGB
  profile: sRGB IEC61966-2.1
  creation: 2010:04:08 11:21:56
  make: NIKON CORPORATION
  model: NIKON D60
  software: Ver.1.00
```

sips is really a hidden gem in Mac OS X.

MACNEWS™

News and information for Apple users.

www.macnews.com


RECOVER YOUR MEMORIES
...before it's too late
LC TECHNOLOGY INTERNATIONAL
PHOTORECOVERY for Digital Media
www.LC-Tech.com/offers | (866) 603-2195

Conclusion

Even if this script *specifically* doesn't help you out, I hope the walkthrough and tear-apart does help. The pattern of processing every directory in a given path is a useful and common one. This fundamental code could live as-is in a bash script called directly from the command-line, or, it could be part of an Automator workflow that uses the Run Shell Script action to call a shell command.

Media of the month: *Winnie the Pooh*—For obvious reasons.

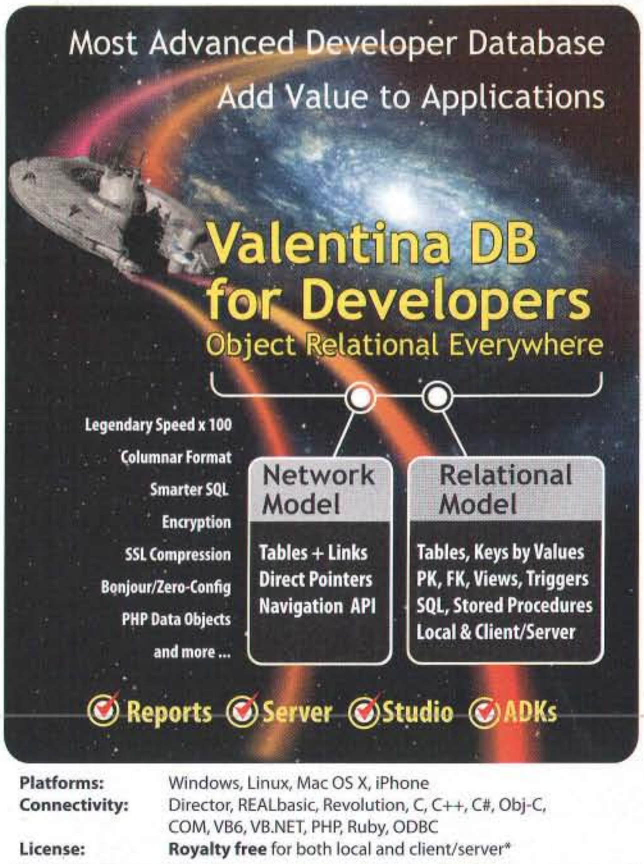
If you're reading this while at WWDC, enjoy the show! This is the first time in many years that I'm not attending. Of course, I'm looking forward to the announcements and surprises that come out of the show and, certainly, to WWDC 2011.

Next month: more bash tips, ticks and idioms.

MT

About The Author

Ed Marczak is the Executive Editor for MacTech Magazine, and has written the **Mac in the Shell** *column since 2004.*



Platforms:	Windows, Linux, Mac OS X, iPhone
Connectivity:	Director, REALbasic, Revolution, C, C++, C#, Obj-C, COM, VB6, VB.NET, PHP, Ruby, ODBC
License:	**Royalty free** for both local and client/server*

www.valentina-db.com

Coming soon......

DocMoto – Document and File Management system for Mac

DocMoto is a complete document and file management system designed specifically for the business OS X user.

Complete with full API's, DocMoto offers the OS X developer an amazing opportunity to develop business critical applications.

For the full story see our website **www.chlsoftware.com**

CHL Software
wwdc2010@chlsoftware.com

docmoto MAC

Configuring Git

Learn how to use the **git-config** command

by José R.C. Cruz

Introduction

In today's article, we learn how to configure the Git tool to suit our revision control needs. We begin with a look at the **git-config** command and its settings files. Next, we explore the many ways we can configure Git. Finally, we give the command a simple graphical interface using Xcode and AppleScript Studio.

Readers are expected to know their way around bash and AppleScript Studio. The Xcode project featured here is available from the MacTech ftp site at ftp.mactech.com/src/mactech/volume26_2010/26.05.sit.

Reasons To Configure

If you recall, your first task after installing Git is to identify yourself to the tool. This is done by entering the following lines at the Terminal session.

```
    git-config —global user.name="your-full-name-here"
    git-config —global user.email="your-email-address-
here"
```

Git uses these settings to mark your changes to a project file that are committed to the repository. These same settings then appear in any project report produced by Git.

Suppose, though, that you want to customize certain aspects of your project repository. Perhaps you want Git to use different colors when it displays a project's status. Or you may want Git to use a different command-line editor. Perhaps you want to prevent accidental changes made to the repository. Or you want to assign shortcuts to certain Git tasks.

The `git-config` command

The **git-config** command defines the overall behavior of the Git tool. It holds its settings in a separate file and retrieves them for various SCM tasks. In fact, many Git commands rely on **git-config** to guide their individual processes.

The command itself takes in five arguments, three of which are optional (Figure 1). The argument **config.ops** sets the operation to perform on a given setting. The argument **config.setting** selects the setting itself.

```
git-config [config·file] [config·type] ¬
  config·ops config·setting [config·value]
```

Figure 1. Syntax of the git-config command.

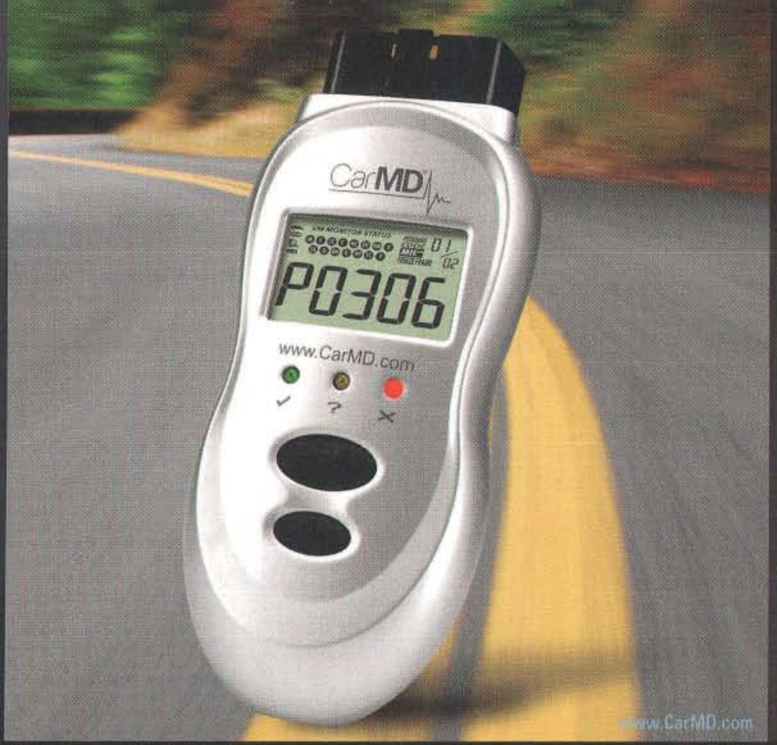
The optional argument **config.file** chooses the location of the settings file. If omitted, Git uses the *project repository* as the default location. The argument **config.type** sets the data-type for the given setting. Git uses this argument to convert the settings value to the correct type. The last argument, **config.value**, sets the value of the setting. Obviously, it is needed only when Git has to create or change a given setting.

Be aware that you could corrupt your repository with your customized settings. So always back up your repository and always test your settings first on a separate non-critical repository.

The settings files

Now Git stores its settings in two separate files (Figure 2). The first file, **.gitconfig**, is an invisible file residing in your home directory. It holds those settings that are common to all project repositories. To use this file, set the **config.file** argument to the **—global** flag.

The second file, **config**, is part of the invisible directory **.git**. This directory is the local repository for the given project. Both directory and file are created when we type **git-init** to prepare the project for Git. To use the config file, simply omit the **config.file** argument from the **git-config** command. But make sure to run git-config *inside* the project directory to access the file correctly.



1 Plug It In!

Use from the safety and comfort of your front seat. Works on all 1996 & newer cars, light trucks, SUVs and minivans.

2 Get The Info!

- Everything OK
- Possible Problem
- Service Required

CarMD makes it easy to quickly identify your vehicle's health.

3 Run A Report!

CarMD.com provides you with empowering information. Walk into any repair shop knowing what needs to be fixed, what doesn't and how much it should cost.

WITH CarMD®, CATCHING HIDDEN ENGINE PROBLEMS IS EASY AS 1-2-3

When you have an important deadline at work or a weekend getaway planned, the last thing you need is for car trouble to slow you down.

In today's cars and trucks, virtually everything's computerized. And even if you do have a degree in Computer Science, it's hard to determine whether that dashboard-warning light is serious or simple. Thanks to the revolutionary CarMD® handheld device and software kit, now virtually everyone who owns a car and a computer can quickly and easily solve automotive repair mysteries.

How it Works

CarMD gives the average driver a way to catch hidden engine problems before the dashboard warning lights come on and they become expensive repairs. Just plug the handheld CarMD device into your vehicle's Data Link Connector (DLC). The DLC is a small port found right under the dashboard on 1996 and newer vehicles – even the newer model hybrids. It's the same place your mechanic plugs in his expensive diagnostic tools. Not sure where to connect? Visit www.CarMD.com to search by year, make and model.

In seconds, CarMD beeps to confirm the test is complete. Then the tool's built-in LEDs let you know how severe the problem is (Green = OK, yellow = possible problem, red = service required). These handy lights are a great litmus test when you're shopping for a used vehicle, or getting ready to hit the road for a long getaway.

CarMD FOR THE LIFE OF YOUR CAR

Mac Universal

To learn more about your car's problem, connect CarMD to your computer using the included software and USB cable. CarMD customers gain free access to an extensive online database (www.carmd.com) that helps diagnose the most likely fix and estimate what repairs should cost down to fair parts and labor in your region.

Coverage

CarMD works on 1996 and newer cars, light trucks, minivans and SUVs – foreign and domestic. It monitors everything that's covered by a vehicle's on-board diagnostic system, which translates to about 80 percent of the systems ranging from a loose gas cap to a catalytic converter failure. Recently launched CarMD Version 3.0 software, which is now compatible to run natively under Mac OS X in both Power PC- and Intel-based Macintosh computers (Version 10.4.4 and newer– including Snow Leopard). Lifetime software and firmware updates mean the CarMD tool you buy today will work on the car or truck you invest in tomorrow.

Special Offer* $88.99

(a $10 savings)

When you buy online at: www.**CarMD**.com

Enter promo code **MacTech2** at checkout

*Offer good through May 31, 2010. **Act now!**

STAY GREEN

CarMD can help keep your car – and your wallet – greener too. The "Check Engine" light often indicates problems with the emissions system. CarMD can diagnose problems that lower your mileage and pollute the environment. Its color-coded LEDs will also tell you if your vehicle is ready to pass its emissions test, so you don't take time off work for a smog check you might fail. CarMD customers gain access to staying healthy reports that include technical service bulletins and safety recalls that can often lead to free repairs.

"If I did not have the CarMD analyzer and computer software I would have wasted approximately 3 hours going to the dealership. As far as I'm concerned, it has paid for itself already."

– RobinJ., N. Ridgeville, OH

To learn more, visit CarMD at:

www.CarMD.com

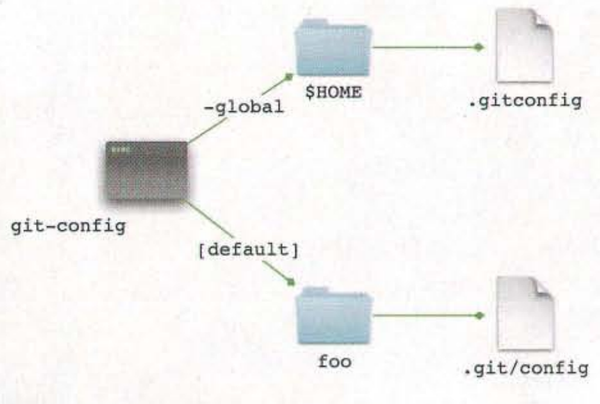


Figure 2. The two settings files.

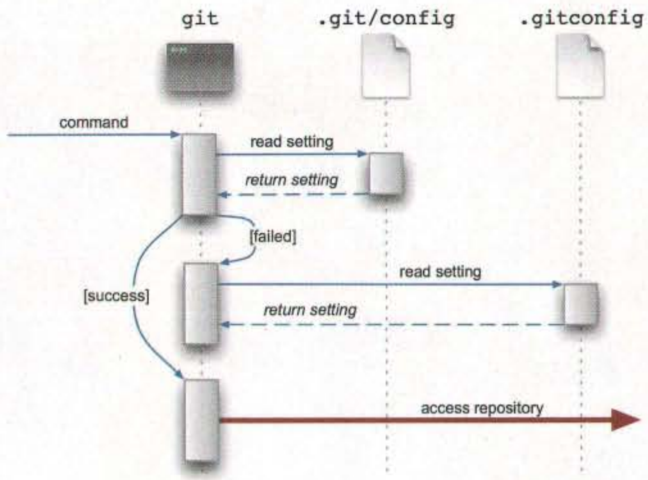


Figure 3. Reading the settings.

Listing 1 shows the contents of a typical settings file. The file stores each setting as a *key/value pair*. It places the key label on the left of the equals sign, and renders the value as a string. Next, the file divides the settings into *multiple sections*. Each section starts with its name enclosed with square brackets, then followed by its share of settings. The end of each section is marked by another section name or by an end-of-file token.

Listing 1. Contents of a settings file.

```
[core]
    repositoryformatversion = 0
    filemode = true
    bare = false
    logallrefupdates = true
    compression = 0
    autocrlf = true
    symlinks = true
[user]
    name = Iosephus Crucis
    email = crucis.iosephus@mail.box.foo
    signingkey = d34db33fd43d
[alias]
    patch = diff -p —color —stat
[clean]
    requireForce = false
```

Each section contains settings that have a *common function*. For instance, the **core** section lists those settings common to all source-control tasks. The **user** section holds settings that are user-specific. And the **clean** section lists those settings that affect garbage collection.

Figure 3 describes how the Git tool uses its two settings file. After Git receives an SCM command, it first checks the repository file for the right settings for the given command. Failing that, it checks the global file for the same settings. If that too fails, Git assumes the *default behavior* when running the command. Failure usually occurs when the file does not have the desired setting or when the file does not exists. Notice that repository settings always take precedence over the global ones. Plus, Git always *expects* the global settings file to exist.

The environment variables

Git also checks two environment variables for the location of a settings file. The variable **GIT_CONFIG** provides the location of the project settings file.

```
export GIT_CONFIG=/Network/Public/Settings/foo_git.settings
```

When this variable is defined, the referred file takes the place of the default **.git/config** in all project repositories. It also overrides the **.gitconfig** global file from the home directory. The second variable **GIT_CONFIG_LOCAL** also points to another project settings file.

```
export
GIT_CONFIG_LOCAL=/Network/Public/Settings/foo_git.settings
```

The referred file also overrides the **.git/config** file inside each project repository. But unlike the **GIT_CONFIG** variable, **GIT_CONFIG_LOCAL** allows access to the global file **.gitconfig**.

Using the Command

Recall the command syntax in Figure 1. To access a setting, first name the setting with the **config.setting** argument. Make sure the name contains both section and key names using a dot-notation syntax. For example, this **git-config** statement assigns a value to the **user.name** key.

```
git-config user.name "Iosephus Crucis"
```

Here, it writes the string **"Iosephus Crucis"** to the settings file **.git/config**. If the file does not have the **user.name** entry, **git-config** will make a new entry as follows.

```
[user]
    name="Iosephus Crucis"
```

To read the **user.name** key, enter the statement with the value omitted.

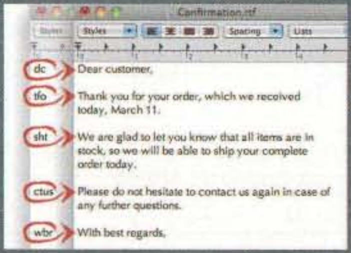
Work less. And finish faster.*

Let us do the tedious work for you.

Typing the same text over and over?

Typinator

types frequently used text for you and auto-corrects your typos.



dc → Dear customer,

tfo → Thank you for your order, which we received today, March 11.

sht → We are glad to let you know that all items are in stock, so we will be able to ship your complete order today.

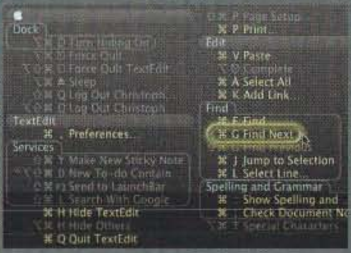
ctus → Please do not hesitate to contact us again in case of any further questions.

wbr → With best regards,

Want to drive your Mac like a Pro?

KeyCue

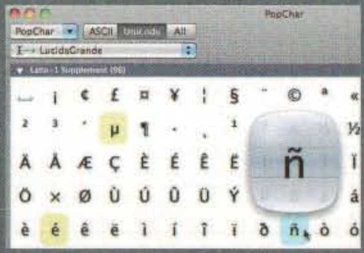
helps you speed up your daily tasks with keyboard shortcuts.



Need special characters in your document?

PopChar

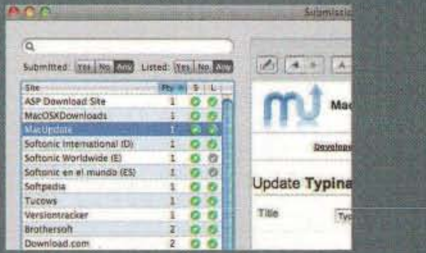
inserts any special character into your document with just two clicks.



Have great software to sell?

Shareware Publisher

submits your software products to leading download sites.



Site	Pty	S	L
ASP Download Site	1	✓	✓
MacOSXDownloads	1	✓	✓
MacUpdate	1	✓	✓
Softonic International (D)	1	✓	✓
Softonic Worldwide (E)	1	✓	●
Softonic en el mundo (ES)	1	✓	●
Softpedia	1	✓	✓
Tucows	1	✓	✓
Versiontracker	1	✓	✓
Brothersoft	2	✓	✓
Download.com	2	✓	✓

www.ergonis.com/mactech

* **Caution:** Usage of Ergonis productivity boosters will revolutionize the way you use your Mac and make you more productive. You will have to look for another hobby!

```
git-config user.name
```

As shown above, the setting name follows the syntax of **section.key**.

These two examples also target the project settings in **.git/config**. To target the global settings in **.gitconfig**, add the **–global** option to the **git-config** statement. Thus assigning a value to **user.name** then becomes as follows.

```
git-config –global user.name "Iosephus Crucis"
```

To read the same setting, the statement is as follows.

```
git-config –global user.name
```

We can also read a setting from the file by setting the **config.ops** argument to **–get**.

```
git-config –get user.name
```

But if we set the same argument to **–get-all**, Git displays the settings from both global and project files, with the global value listed *first*.

```
git-config –get-all user.name
```

To remove an existing setting from the settings file, set the **config.ops** argument to **unset**. For example, this statement removes the **user.name** entry from the project file.

```
git-config –unset user.name
```

But Git will not remove the section after it has removed that section's last setting.

Working with sections

Next, the **git-config** command has the ability to manage the sections of settings within a given file. Suppose, for example, we need a custom section for settings that are project-specific. To create this section, simply use the command as follows.

```
git-config project.host "OS X 10.5.1"
```

Here, we add a new section named project to the **.git/config** file. Under that section, we add a host setting with its value set to the string **"OS X 10.5.1"**. To read the custom setting, simply name the setting and use the **–get** operation as follows.

```
git-config –get project.host
```

To list all the sections and settings, use the **git-config** command with a **list** operation.

```
git-config –list
```

To list only those settings that belong to a specific section, pipe the command output to **grep** as follows.

```
git-config –list | grep "project\."
```

Here, the **grep** tool filters out only the settings under the custom section **project**.

To rename a specific section, use the **git-config** command with a **–rename-section** ops.

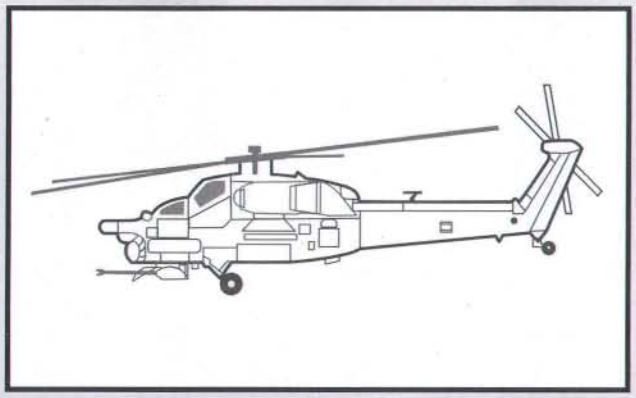
```
git-config –rename-section project foo
```

In the above example, Git searches for the custom section

Left Brain or Right Brain --- Choosing EazyDraw is a No-brainer

Left Brain: Logic, details, math and science, reality, forms, strategies, words and language.

Right Brain: Feeling, big-picture, imagination, symbols, images, perception, fantasy, possibilities.



Whether your ideas float off the page or need to claw their way to reality, EazyDraw is with you from the big picture to the details.

Technical Drawings
Charts & Diagrams
Text Layout
House Plans

eazydraw®
have fun drawing on OS X

Illustrations
Logo Design
Web Graphics
App Icons

www.eazydraw.com or call +1 608.444.5245

Vector Drawing App for OS X

CAMTASIA
[PRONOUNCED cam•TAY•zha]

INTRODUCING
THE ALL-IN-ONE
SCREEN
RECORDER
DESIGNED
FOR MAC

Record screen activity
Edit your content
Add effects
Share your video

Download the free trial and check out our introductory pricing today at: **www.camtasiamac.com**

Camtasia:mac
TechSmith

project in the .git/config file. If the section exists, Git then gives it the new name foo. This also moves all the settings under the project section to the new foo section. To test if the move is successful, use the git-config command with a –get operation.

```
git-config –get foo.host
```

To remove a section from the settings file, use git-config with a –remove-section operation.

```
git-config –remove-section foo
```

The above example not only removes the foo section, it also removes the settings under that section.

When the command fails

Of course, there are cases where a git-config statement fails. One such case is where the settings file itself is *invalid*. The file may be corrupted or it may be rendered in some unknown format. It is even possible that the file has the wrong text encoding, the correct one being UTF-8. If this is so, git-config will return fatal error or it will return a null result. So far, the best way to resolve the problem is to delete the file with an rm command. This allows Git to create an empty settings file wherein to store its settings.

```
rm -f .git/config
```

Another case is when the file is in a *read-only state*. This happens when the file sits inside a read-only directory. Git will be unable to read its settings from the file, or will it be able to write or update them. To correct this problem, use the chmod command to give the directory the correct set of permissions. As this is an admin task, make sure to use chmod in a sudo statement.

```
sudo chmod 666 .git
```

Then provide the correct administrative password when prompted.

The git-config command also "fails" when the named setting is either *incomplete* or *non-existent*. Consider the following statement.

```
git-config –get project.foo
```

If the settings file does not have a project section, the above statement returns a null result. The same also happens if the file does have a project section, but not the key foo. Now consider this statement.

```
git-config foo test
```

This one appears to set the key foo to a value of 'test'. But Git responds with an error message stating that the section foo is absent from the settings file. Note that each action produces a completely result.

One more way for git-config to fail is when the settings file has multiple entries of the same key. Assume the file has two entries of the key project.foo. Reading that key results in Git returning an error message

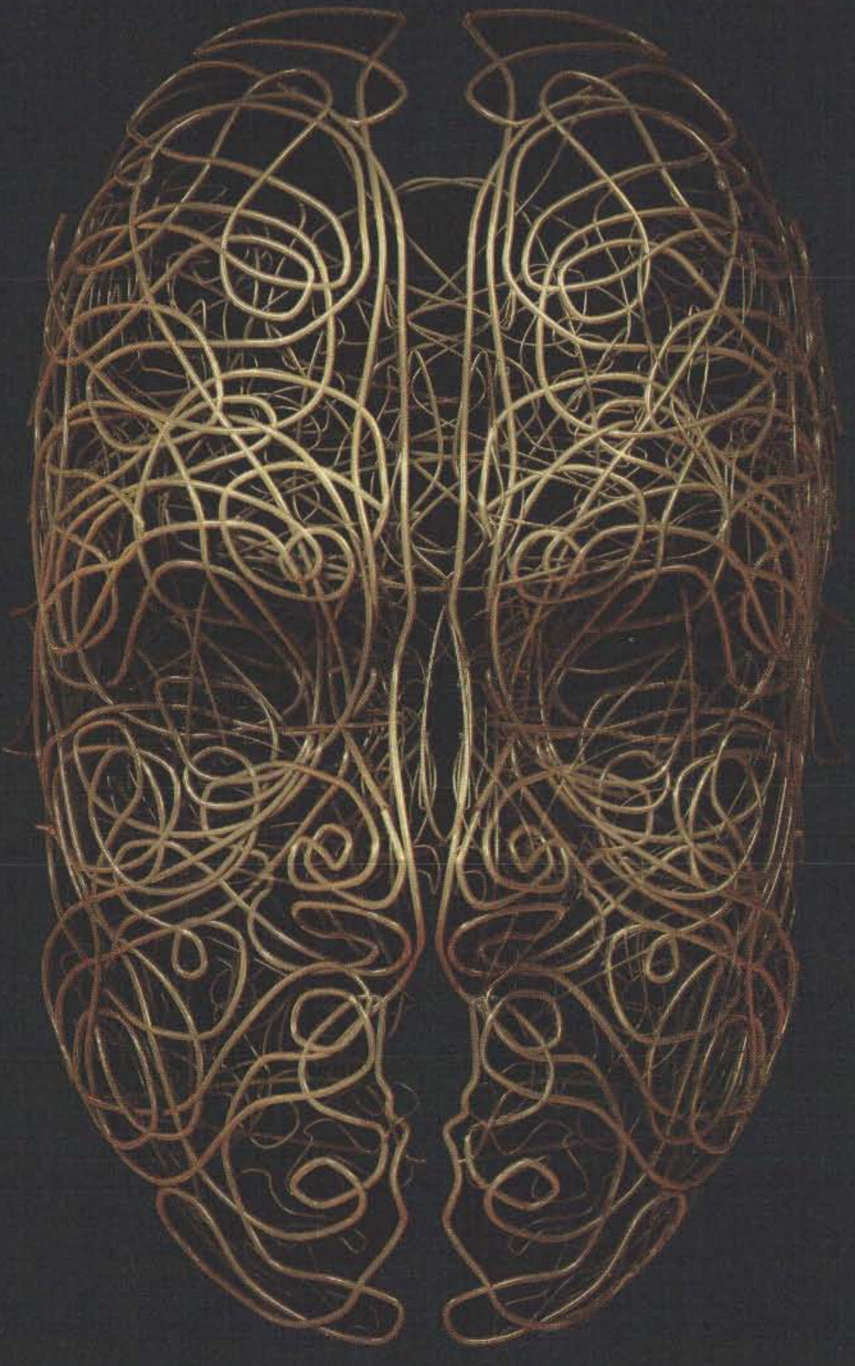
```
git-config –get project.foo
error: More than one value for the key project.foo
```

More than just a pretty face.

Flesh out the concept with 3D characters

Let the technical go and your creativity flow

Conquer the challenge in record time



POSER® PRO 2010

Professional 3D Figure Design & Animation

Poser Pro 2010 is the most efficient way to include 3D figures in your next creative project. With over 2.5GB of ready to use assets, pose and animate beautifully sculpted 3D characters, without needing to model and rig them. Plug-in to 3ds Max, Maya, CINEMA 4D and LightWave, or use COLLADA to bring Poser content into your favorite production tools. It's easy. It's fast. It's creative. It's Poser Pro 2010. **Visit poser.smithmicro.com**

smithmicro software

Setting that key to a new value causes Git to return a warning message.

```
git-config project.foo 789
Warning: project.foo has multiple values
```

The same warning also appears when trying to remove that same key. Currently, the only way to correct this problem is to open the errant file in a text editor and then delete the duplicate entries.

A Survey of Settings

Let us now explore the many ways we can configure our Git setup. In this article, we will focus only on settings that affect an everyday SCM session. For a complete list of possible settings, consult the online Git manual in the references.

User information

So far, we changed two **git-config** settings to identify ourselves to our Git setup. To identify by name, we changed the **user.name** setting.

```
git-config user.name "Iosephus Crucis"
```

To identify by e-mail address, we changed the **user.email** setting.

```
git-config user.email "iosephus.crucis@mail.box.foo"
```

Git then uses the above settings to sign our changes to the project. It also uses both settings to produce a GPG key for tagging. But did you know that we could supply our own unique key for tagging? To do so, pass the key to the **user.signingkey** setting.

```
git-config user.signingkey deadbeef
```

Unlike the previous two settings, we can change the **user.signingkey** setting only in the project settings file.

We can even override our identity settings with two environment variables. To override the **user.name** setting, use the **GIT_AUTHOR_NAME** variable.

```
export GIT_AUTHOR_NAME='Crucis, Iosephus'
```

To override the **user.email** setting, use **GIT_AUTHOR_EMAIL**.

```
export GIT_AUTHOR_EMAIL='crucis.iosephus@mail.box.foo'
```

Git will use these two strings to sign project changes committed from that point on. But earlier changes will remained signed with the older strings.

Repository control

We can also use **git-config** to control how Git manages our project repository. For instance, when Git adds a text file to the repository, it preserves the original end-of-line (**EOL**) tokens of that file. Windows use **CRLF** (**0x0d0a**) for their text files, while POSIX uses just **LF** (**0x0d**). Suppose you want to support only the POSIX token. To do so, set the **core.autocrlf** setting to **TRUE**.

```
git-config core.autocrlf true
```

With the above setting set, Git converts a committed file's **EOL** token to the POSIX version. And during checkout, Git updates each file's **EOL** to the right token for the sandbox's platform.

Offshore web hosting & domain registration with advanced security features.

Located in the Bahamas, NameVault offers you the increased security and benefits your website needs. With direct connections to all major networks, NameVault is the ICANN accredited leader in domain name registration and hosting solutions.

namevault.com®
Secure Your Domain

Next, Git treats a file's name or path as a quoted string when the latter contains unusual characters. Such characters are those with ASCII numbers greater than `0x80` (`DEL`). To force Git to interpret these characters verbatim, set the **`core.quotePath`** setting to `TRUE`.

```
git-config core.quotePath false
```

This setting ignores certain characters like double quotes (`0x22`), backslashes (`0x5c`) and control characters (`0x01` to `0x1f`).

Git also treats *alias files* as valid repository items. When we commit an alias to the repository, Git stores the alias itself, not the file or directory that it refers. To disable this behavior, write `FALSE` to the **`core.symlinks`** settings.

```
git-config core.symlinks false
```

Doing so tells Git to store the files or directories to which the alias points. This allows us to use Git on FAT volumes, which do not support aliases. And it helps us identify broken aliases, aliases that do not point to a valid file or directory.

When Git adds files to a project repository, it *compresses* each file to conserve space. The algorithm used is `DEFLATE`, which is supplied by the system library file `zlib`. Now Git lets `zlib` compress the files using its default factor. To change this, write the new compression factor to the setting **`core.compression`**.

```
git-config core.compression compression-factor
```

The argument *`compression-factor`* is an integer ranging from 0 to 9. A value of 9 chooses *maximum compression* at the cost of increased access time. A value of 0 *disables* compression, while a value of -1 choose the *built-in default*.

Git also has the ability to remove unused and unwanted files from a project repository. This helps reduce repository size and shorten access time. The command to invoke this feature is the `clean` command.

```
git-clean
```

But if this command is used too often, it may remove important files or invalidate the history log. So to restrict its use, set the **`clean.requireForce`** setting to `TRUE`.

```
git-config clean.requireForce true
```

With the above setting, just typing `git-clean` will not work. If we want to use the `clean` command, we must add a `-f` option.

```
git-clean -f
```

Making aliases

One challenge faced by many new users is the plethora of options available to each Git command. Knowing how each option works and when to use one can be confusing at times. Plus, typing out these options frequently and repeatedly can be tedious, prone to errors. This is why Git lets us assign an *alias* to a command statement.

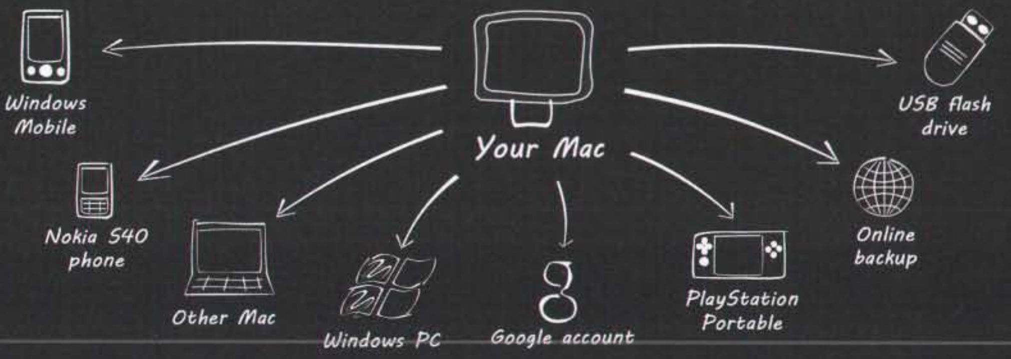
For example, consider this `git-diff` statement.

```
git-diff -p —color —stat foo.txt
```

This one tells Git to compare the file `foo.txt` with its latest revision in the repository. Git then displays the differences in color and using a unified format. It also displays the basic statistics

SYNCMATE

has the power to make all your devices work in sync with your Mac



freeware

Free SyncMate can keep your Address Book and iCal up-to-date on all devices you use daily.

Get more with SyncMate Expert: sync your mail, music, photos, videos, To Dos, bookmarks and more!

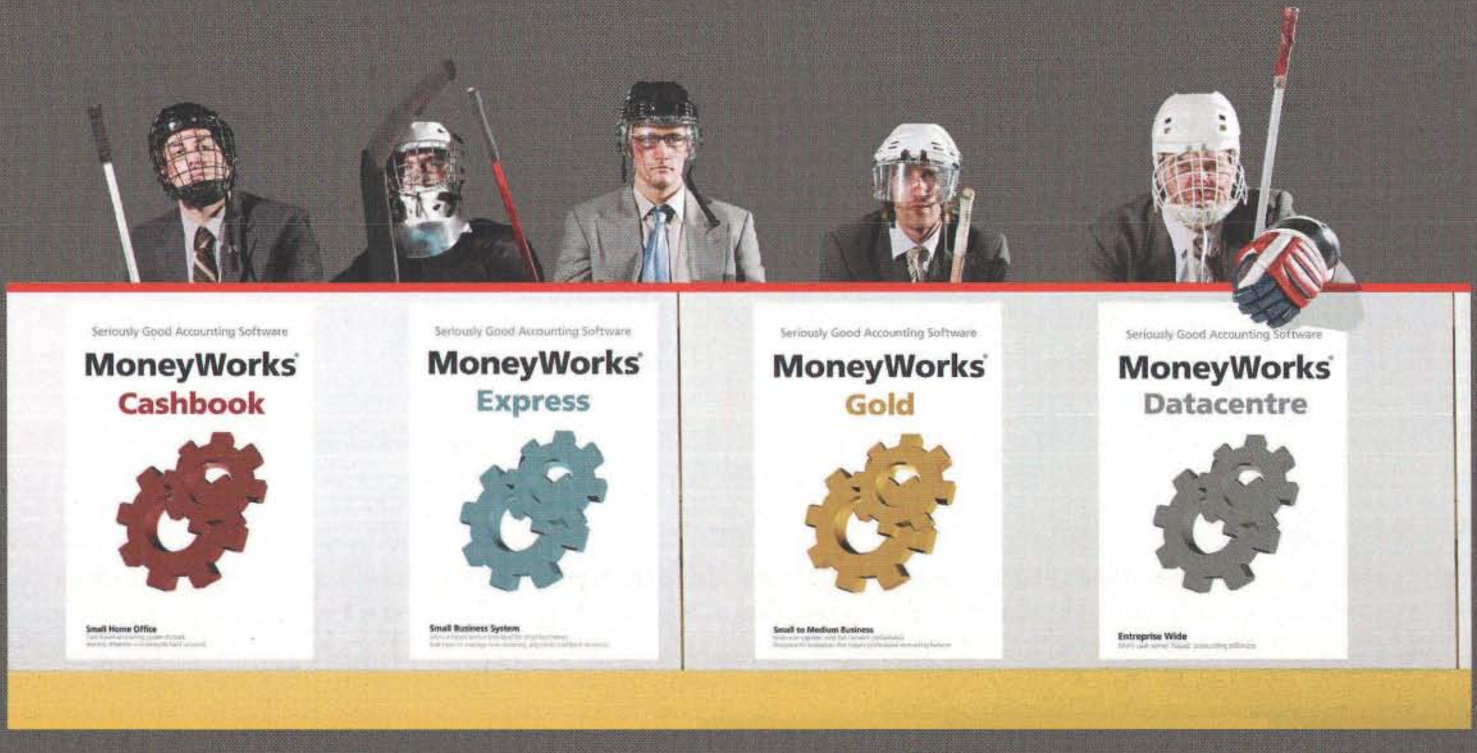
http://eltima.com/syncmate Coupon code (20% off): SYNCMATE-20

MoneyWorks®

Seriously Good Accounting Software

Our customers tell us that our Canadian support team is helpful and polite.

They are, if you keep them off the ice.



Give them a call at 800-818-6955. It's free.

Or if it is after office hours when they are out playing hockey, send an email to support@moneyworks.ca. Emails are also free.

Download a trial version at moneyworks.ca/us/downloads regarding the two files. To assign an alias to this statement, use the **git-config** command as follows.

```
git-config alias.patch 'diff -p —color —stat'
```

This adds a new entry named **alias.patch** to the project repository settings. The aliased command and its options appear as a *string enclosed in single quotes*. To use the alias, type the following statement at the shell prompt.

```
git patch foo.txt
```

Notice the "command" **patch** is the same name we gave to the above alias.

Edits and reports

We can even use **git-config** to adjust some aspects of our commit and report sessions. For instance, when Git commits our changes to a project file, it launches **vim** wherein we describe those changes. To use a different editor, enter its name into the **core.editor** setting.

```
git-config core.editor nano
```

Here, we told Git to use the **nano** tool as our editor. The next time we commit a file, Git will launch **nano** to accept our commit message. Make sure the tool exists and its location mapped by the **PATH** variable. Otherwise, Git will revert to the **vim** tool. We can also name the desired editor using the **EDITOR** variable as follows.

```
export EDITOR /usr/bin/nano
```

When Git displays long outputs such as a project's revision log, it divides the output into separate pages using the command-line tool **more**. To switch to a different tool, pass the tool's name to the **core.pager** setting.

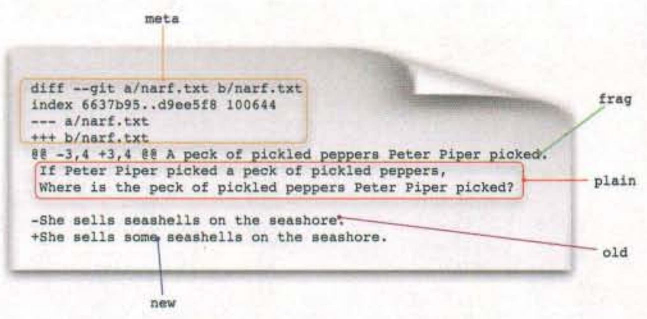
```
git-config core.pager ipage
```

In this example, we have Git pipe its output to the custom tool **ipage**. We can also affect the same action by setting the **GIT_PAGER** variable as follows.

```
export GIT_PAGER /Library/usr/bin/ipage
```

The above statement assumes that the tool sits inside the **/Library** directory.

Another use for the **git-config** command is to enable *colored reports*. By default, Git displays its reports in monochrome, usually black text on white. But colored text can make long reports easier to read and help identify trouble spots. How Git colors its reports depends on the report content. Consider the **git-diff** report for example (Figure 4). A typical report has five distinct regions or **slots**. The first slot (**meta**) is an overview of the report. It lists the files and repository entries affected by the changes. The **frag** slot gives a summary of the changes. It describes the number of lines to be deleted or added should we decide to commit our changes. The third slot (**plain**) lists the first two lines that precede our changes. The **old** slot is the line before the change, the **new** slot the line after the change.



```
diff --git a/narf.txt b/narf.txt
index 6637b95..d9ee5f8 100644
--- a/narf.txt
+++ b/narf.txt
@@ -3,4 +3,4 @@ A peck of pickled peppers Peter Piper picked.
 If Peter Piper picked a peck of pickled peppers,
 Where is the peck of pickled peppers Peter Piper picked?

-She sells seashells on the seashore.
+She sells some seashells on the seashore.
```

Figure 4. Parts of a diff report.

To get a colored diff report, set the **color.diff** setting to **TRUE**.

```
git-config color.diff true
```

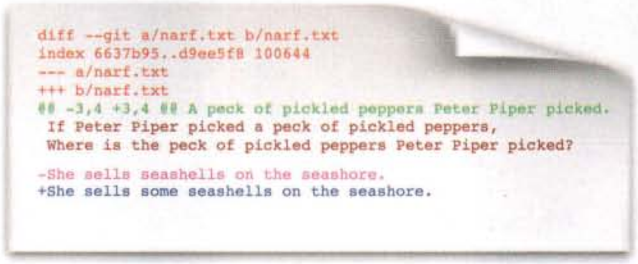
To assign a color to each slot, append the slot name to the **color.diff** setting. Then pass the color to each slot as follows.

```
git-config color.diff.meta yellow
```

Here, we set the **meta** slot to *yellow*. To set the **plain** slot to red, modify the above statement as follows.

```
git-config color.diff.plain red
```

Other possible colors include *black*, *green*, *blue*, *magenta* and *cyan*. Once we set the desired colors, typing **git-diff** will give a report similar to that in Figure 5.



```
diff --git a/narf.txt b/narf.txt
index 6637b95..d9ee5f8 100644
--- a/narf.txt
+++ b/narf.txt
@@ -3,4 +3,4 @@ A peck of pickled peppers Peter Piper picked.
 If Peter Piper picked a peck of pickled peppers,
 Where is the peck of pickled peppers Peter Piper picked?

-She sells seashells on the seashore.
+She sells some seashells on the seashore.
```

Figure 5. A colored diff report.

Similar settings are also present for the **git-branch** and **git-status** reports. To enable colored branch reports, set the **color.branch** setting to **TRUE**.

```
git-config color.branch true
```

To assign a color to a specific branch slot, say for instance **remote**, pass the color to the slot as follows.

```
git-config color.branch.remote yellow
```

And to enable colored status reports, set the **color.status** setting to **TRUE**.

```
git-config color.status true
```

A complete list of slot settings for these two reports can be found in the online Git manual (see references).

APPRIVER CLIENTS NEVER SEE THIS

Error: Mailbox Full

You have exceeded the storage limit on your mailbox. Delete some important emails or contact your server administrator who is on vacation right now. My neighbor's kid is really good with computers. Maybe he could help you out when he gets home from school. Or call AppRiver to solve this problem now.

Call AppRiver

FREE 30-Day Trial Sign-up NOW

What's New:

Native support for Microsoft® Exchange allows users to integrate their Exchange data into Mac's **Mail, iCal** and **Address Book** applications.

Unlimited Mailbox Storage
No need to manage your mailbox storage quotas.

Spam & Virus Protection
Comprehensive protection against unwanted email.

Entourage & Mac Mail Optimized
Email Performance you would expect on your Mac!

Better Mobility
Synchronize your iPhone with our FREE ActiveSync service, the fastest, most reliable synchronization service available.

appriver® The Messaging Experts™

www.appriver.com/mac

sales@appriver.com

(866) 223-4645

MACTECH® domains

Register

Get your .COM or any other domain name here!

FREE with every domain:

- **FREE! Starter Web Page**
- **FREE! Getting Started Guide**
- **FREE! Complete Email**
- **FREE! Change of Registration**
- **FREE! Parked Page w/ Domain**
- **FREE! Domain Name Locking**
- **FREE! Status Alert**
- **FREE! Total DNS Control**

Just visit

www.mactechdomains.com

to register for your domain today!

Starting at $1.99

when a non-domain name product is purchased. Limitations apply.

Summing Up

The `git-config` command gives us the means to prepare and configure our Git setup. With it, we identify ourselves to the project repository and to any users sharing that same repository. We can change how Git manages the repository and how it presents its reports. We can assign shortcuts to help simplify often-used tasks. We can even enable safeguards to deter accidental changes to the repository.

And that ends today's study of the open-source Git tool. But stay tuned to this spot as we learn more about the tool and its capabilities. Next month, I'll present a GUI tool that can alter git-config settings. Until then, I bid you good day.

Bibliography and References

Scott Chacon (editor). *The Git Community Book*, pp. 21–22. GitHub [Online]. Available: http://book.git-scm.com.

Johannes Schindelin. "git-config". *The Git Manual*, The Linux Kernel Archives [Online]. Available: http://www.kernel.org/pub/software/scm/git/docs/git-config.html.

Bor Kraljic. *The Git Guide*, SourceMage Wiki [Online]. Available: http://www.sourcemage.org/Git_Guide.

MT

About The Author

JC is a freelance engineering writer from North Vancouver, British Columbia. He writes for various publications, covering diverse topics such as AppleScript, REALbasic, Python, and Cocoa. He also spends quality time with his foster nephew. You can reach JC at anarakisware@gmail.com.

BUNDLED PHONE & INTERNET SERVICE

FROM $459 FLAT RATE

Dynamic Allocation T-1
Up to 16 Business Lines
Unlimited Local Service
Unlimited Site to Site Calling
2,000 Minutes of Long Distance or Toll Free

Voice Mail, Call Forwarding, 3-Way Calling, Call Hold, Pickup and Transfer, Call Waiting, Last Number Redial, DID, and DOD, Caller ID and more!

www.lowcostdialing.com

800-906-8686

From print to web in a Flash. Work your magic with QuarkXPress® 8

Quark

So you're a master of print design? Time to step it up a notch. Use your existing QuarkXPress skills to design for the Web and bring your creations to full interactive glory – without having to learn Flash or coding. The intuitive design interface of QuarkXPress 8 opens a world of new possibilities. Increase your productivity and offer your clients more (both print and Web), right out of the box.

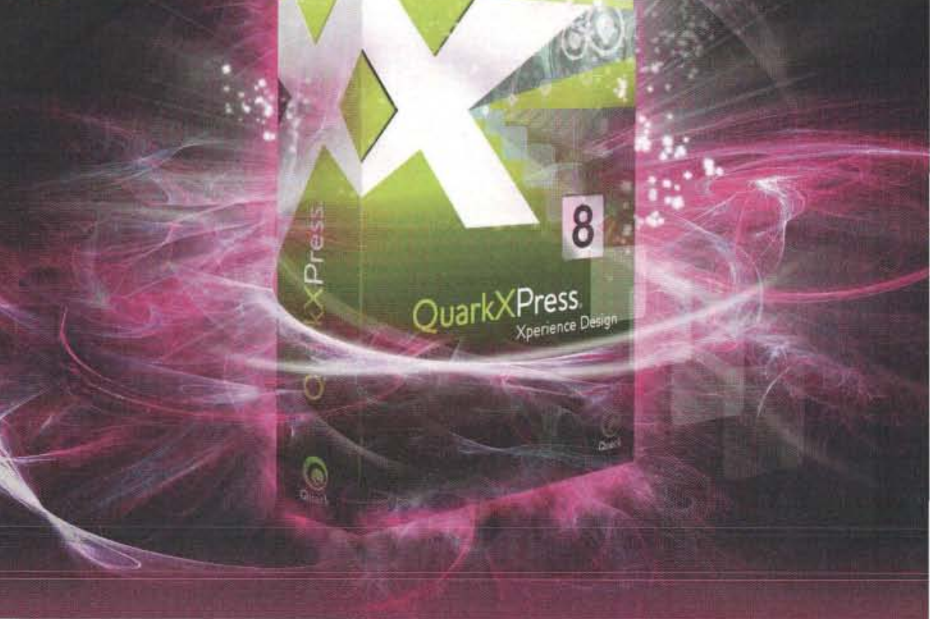
But, there's more to this box of tricks than meets the eye. Buy or upgrade to QuarkXPress 8 today to access $750+ of Exclusive Flash Resources for **FREE** to make your designs shine:

- Hundreds of exclusive, fully editable Flash assets
- Web templates, animations and video players
- Flash tutorials, eSeminars and educational resources

Upgrade NOW for just $299 (before tax) or BUY for $799 (before tax)

Unleash the magic of Flash in QuarkXPress 8 visit www.quark.com/magic

Purchase QuarkXPress® 8 today and access $750+ worth of Exclusive Flash Resources for **FREE**



Find and manage your Mac and PC computers with Absolute® Software. From the largest corporations to your home office, our Computrace®, Absolute Manage and LoJack® for Laptops solutions help you improve data protection, simplify computer lifecycle management and recover stolen computers.

For a FREE trial of Absolute Manage visit:

www.absolute.com/rediscover

Absolute®Software

HOW CAN YOU MANAGE WHAT YOU CAN'T FIND?

Rediscover your computer fleet with Absolute Software.

- Increase auditing accuracy
- Lower compliance risks
- Minimize security risks
- Reduce total cost of ownership

The absolute best way to track, manage & protect your digital world.

WWDC For Enterprise Admins

What to do at WWDC 2010

By Greg Neagle, MacEnterprise.org

MacEnterprise.org
Mac OS X enterprise deployment project

No IT Track?

Apple's World Wide Developer Conference, or WWDC, has grown and changed dramatically in the last few years. Pre-iPhone, the conference had tracks for Mac OS X developers and Mac OS X administrators. The track for administrators was known as the "IT Track", and I had the privilege and pleasure of presenting at WWDC in 2007 and 2008. Post-iPhone, each year the number of iPhone OS sessions increased, and in 2008 and 2009, there were essentially three tracks: Mac OS X, iPhone OS, and IT. Due partially to the excitement and popularity of the iPhone and its App Store, WWDC 2009 sold out in a month.

This year, Apple introduced and shipped the iPad, a third device based on the iPhone OS. In the spring, Apple announced iPhone OS 4, with lots of tasty new features. So it shouldn't have been surprising that when WWDC 2010 was announced, the number of iPhone OS sessions had grown once again. But to make room for more iPhone OS sessions, something had to give: for the first time in many years, there is no IT track at WWDC.

This made no difference as far as the popularity of the conference: this year, WWDC sold out in eight days. Clearly, iPhone OS developers are eager for all the information they can get – and who can blame them? The iPhone/iPod touch/iPad have enabled developers to successfully sell new and wonderful applications for Apple's iPhone OS hardware. The iPhone OS platform is growing at an incredible rate, and that's where all the excitement is today.

Okay, Now What?

So you are a systems administrator or engineer responsible for Mac OS X machines. There's been an influx of iPhones (and maybe even iPads) in your company, and naturally, you are being asked to deal with those as well. You are attending WWDC 2010 this year. But since there is no IT track, what should you do?

Keynotes

There are often multiple keynote addresses at WWDC. Apple almost never releases information on the contents of these keynote presentations ahead of time, so it's impossible to say if they will be of specific interest to people looking for "IT" information. But they are almost always interesting (and sometimes entertaining!) in their own right. Based on the topics of the last few WWDC keynotes and the focus of this year's WWDC, it's not too much of a stretch to guess that the main keynote will be iPhone OS 4-related and might even introduce new iPhone hardware. The "State of the OS" keynote, typically in the afternoon of day one, has traditionally been a good session to learn about the future of Mac OS X. In 2008, for example, this is where Snow Leopard was first announced and described. Who knows what will be discussed this year, but I wouldn't hold my breath for information about Mac OS X 10.7.

Sessions and Labs

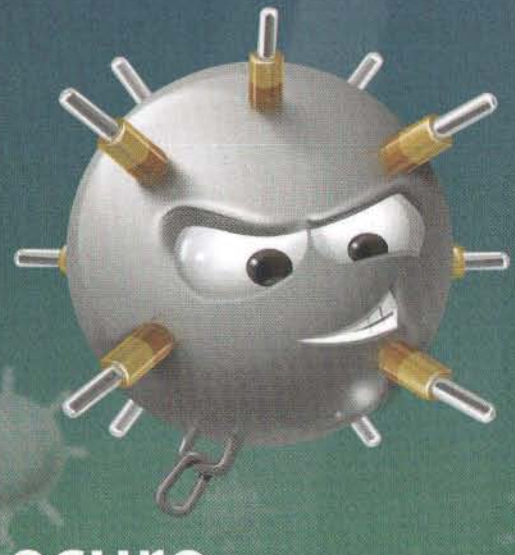
Even with no formal IT Track, there are a few sessions and labs that might be of interest to an enterprise administrator. This column was written about a month before WWDC, so the sessions and labs that have been announced may change for the actual event. But based on what has been announced so far, here are some ideas:

Managing Mobile Devices

If you are now, or will be responsible for supporting iPhone OS devices in your organization, the "Managing Mobile Devices" session should be of interest. From Apple's session description, this session will focus on the new Mobile Device Management architecture and new Over-The-Air (OTA) capabilities available with iPhone OS 4. Of particular note is the new ability to deploy configuration profiles and in-house applications over-the-air. Prior to iPhone OS 4, deploying in-

ProteMac
www.protemac.com

Protemac
NetMine

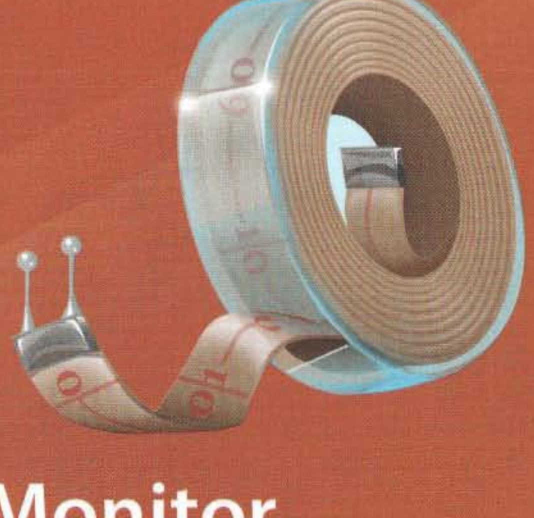


Secure
your network

Protects against all types of malware,
visualizes network activity,
keeps traffic details.

Mac Universal

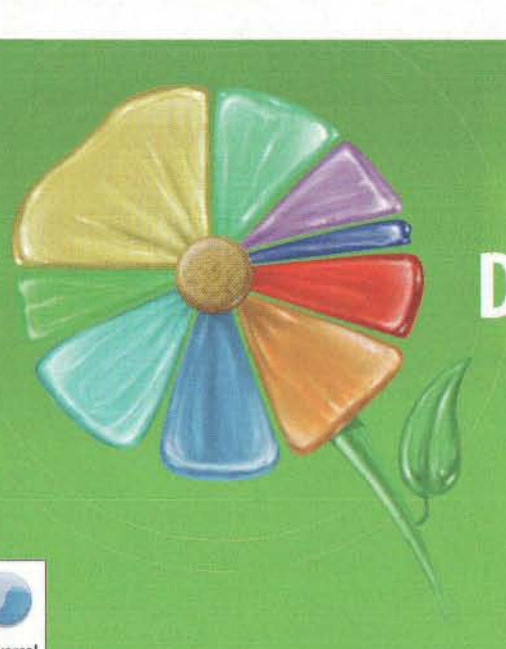
Protemac
Meter



Monitor
your network

Controls network usage, traffic and
safety, measures network traffic,
provides statistics.

ActyMac



ActyMac
Duty Watch

Monitors your employees'
monitors.

Mac Universal

www.actymac.com

house applications required connecting a device to iTunes on an Mac or PC. Over-the-air deployment should make managing iPhone devices in an enterprise much easier.

Along with the "Managing Mobile Devices" session, Apple is also offering a "Mobile Device Management" lab, where you can interact with Apple engineers and get answers to your organization-specific iPhone OS management and deployment questions.

If you are not currently supporting iPhone OS devices in your environment, you may want to attend the "Managing Mobile Devices" session and lab anyway. It seems inevitable that the new computing paradigm hinted at by the iPhone and iPod touch, and now evolved as part of the iPad, will have a growing influence on computing as a whole, even in the enterprise. You owe it to yourself to be ahead of the game and see where things may be going in the future.

Delivering audio and video via the Web

You may manage or support web servers that serve audio and/or video. Many educational institutions make use of Podcast Producer or similar technologies to make lectures and presentations available on-demand via the web. Other organizations may make training materials available in this manner, and some have other internal uses for web-based audio and/or video. If any of these scenarios describe you, there are several sessions of possible interest.

"Advances in HTTP Live Streaming" is a session on deploying live and on-demand video streams using a standard web server. You'll learn how to support streams for both desktop systems and mobile devices like the iPhone.

There will also be two sessions on "Delivering Audio and Video Using Web Standards", which cover using HTML5 and CSS3 to deliver media to Mac, Windows, and iPhone OS clients. While this moves from a strictly administrative focus to more of a web developer focus, it's not uncommon for systems administrators or engineers to have to do some web development.

More web development

Along the same lines, there are several other sessions of potential interest to web developers, including sessions on adding support for touch and gesture events to web pages and using SVG (Scalable Vector Graphics) and HTML5 to create and display graphs, tables, charts, and other dynamic information displays. There are also sessions on using Safari's developer tools to assist in creating, debugging, and optimizing web sites for desktop and mobile devices, and several labs where you can get face time with Apple engineers to answer your questions about all of the above topics.

Lunchtime Speakers

While as of this writing, no specific lunchtime speakers have been announced, take the time to check these sessions out. In most cases, these speakers are non-Apple people who are using Apple technologies in their organizations. Often there

Also available on Blackberry & Windows Mobile

Back Together Again.

Integrating calendar events and personal tasks into one easy to use App. The way it should be.

Pocket Informant

Sync is here *Pocket Informant, the iPhone's most versatile calendar/tasks Personal Information Manager (PIM) sets itself apart from the rest by providing the ability to sync to the other systems you hold dear. Now syncing with Google, Toodledo, Outlook, and Mac OS X.*

(note: Desktop Sync information available at our website)

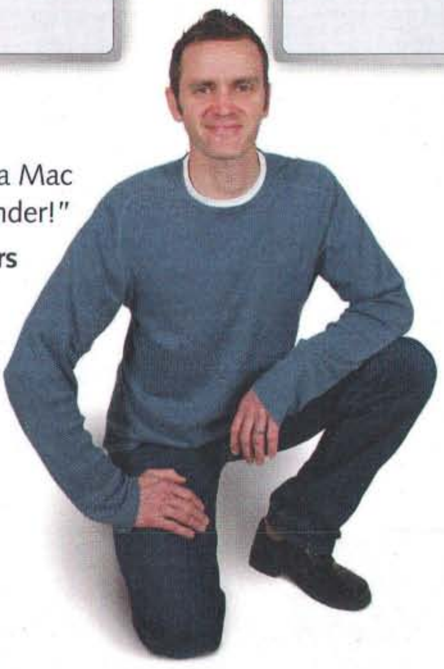
www.pocketinformant.com | 2010 ©

Google
Toodledo

Type Less. Create More.

Effortlessly insert frequently-used words, phrases and graphics, by typing short abbreviations. Great for email, blogging and more!

`thx`	Thanks for your interest in our creative services.
`chk`	Check out our portfolio at: **www.lapis-design.com**
`mysig`	Michael Fetters *Lapis Design*



"I can't work on a Mac without TextExpander!"

Michael Fetters
Mac user

textexpander

Now Available!
TextExpander touch™ for iPad, iPhone & iPod touch

Integrate TextExpander touch in your apps with our free SDK:
www.smileonmymac.com/sdk

Smile on my mac

disclabel PDFpen pagesender textexpander is much to be learned from people using Apple solutions in "real-world" situations.

The "Hallway Track"

Many past WWDC attendees would tell you that the most valuable sessions at WWDC are the impromptu discussions that occur in the hallways between the official sessions. WWDC gathers together a large number of technical people with similar interests, so it should come as no surprise that you may encounter other people with similar issues as you, or ones who've solved an issue you've encountered. With the absence of the IT Track, I suspect the "Hallway Track" will take on added significance this year. Hang out after formal sessions and see what conversations develop. Meal breaks are another excellent opportunity to strike up conversations with similar-minded administrators.

Outside Events

Finally, during the week of WWDC, there are usually outside events of interest to enterprise administrators. AFP548.com often hosts a party one evening, and this attracts both enterprise administrators and enterprise vendors. It's an excellent opportunity for discussing problems and solutions with other Mac OS X administrators. MacEnterprise.org also sometimes hosts an informal get-together at a local restaurant. Neither organization has announced anything for WWDC as of this writing, so be sure to check their websites for current information. It's also likely that one or more enterprise vendor will have some sort of event – these can also be valuable.

Don't overlook the possibility of organizing something yourself. If there is a website or mailing list you frequent that has like-minded people, you can propose an informal meeting – dinner, perhaps, or maybe just getting together for a drink and a chat. The MacEnterprise mailing list is one obvious place to start, but there are many others as well.

Conclusion

While this year's WWDC may have less to offer enterprise administrators than years past, if you are currently supporting or planning to support iPhone OS devices in your organization, you'll probably find sessions of interest. But since a lot of the value of WWDC comes with interacting with other WWDC attendees, it's likely that you'll be able to find a fair amount of value outside of the traditional sessions and labs, even if few of the formal sessions are directly relevant to your organization. Good luck, and don't forget to have some fun!

MT

About The Author

Greg Neagle is a member of the steering committee of the Mac OS X Enterprise Project (macenterprise.org) and is a senior systems engineer at a large animation studio. Greg has been working with the Mac since 1984, and with OS X since its release. He can be reached at gregneagle@mac.com.

Lithium 5.0 New

Network, Server and Storage Monitoring.

Xserve, VTrak, Active Storage, Xsan
SNMP support for everything else.

+ Third Party MIB Support
+ Custom Service Checks

lithium5.com

lithium Corp.

/n software

www.nsoftware.com >

The Best-Selling Internet Communications, Security, and E-Business Components, Now Available On:

Mac OS X and iPhone!

COCOA FRAMEWORKS FOR INTERNET COMMUNICATIONS

IP*Works! eliminates the complexity of Internet development providing easy-to-use, programmable components that facilitate tasks such as sending E-mail, transferring files, managing networks, browsing the web, consuming Web Services, etc.

Internet Communications

- **IP*Works! - [Core Framework]**

A comprehensive framework for Internet development. The core building block for most /n software products.

Components - IPMonitor, MX, REST, NetCode, RSS, NNTP, SMPP, POP, Rexec, Rshell, Syslog, SMTP, WebDav, SOAP, XMPP, Telnet, Ping, TFTP, FilerMailer, UDPPort, HTMLMailer, WebForm, NetClock, WebUpload, RCP, Whois, FTP, XMLp, HTTP, SNPP, IMAP, MIME, IPInfo, IPDaemon, IPPort, NetDial, LDAP, ICMPPort, MCast, TraceRoute

Network Management

- **IP*Works! Secure SNMP**

A comprehensive toolkit for building Secure SNMP-based agent and manager applications including advanced SNMPv3 security features, trap handling, and ASN-1 MIB compilation.

Components - SNMPAgent, SNMPMgr, SNMPTrapMgr, MibBrowser

File & Streaming Compression

- **IP*Works! Zip**

Suite of easy-to-use, fast, effective components for compression and decompression with advanced features including self-extracting archives, industrial strength AES encryption, and Zip64 archives.

Components - GZip, Tar, Jar, Zip, ZipStream

Secure Connectivity & E-mail Confidentiality

- **IP*Works! SSL**

SSL-enabled versions of the components in the core IP*Works! package.

Components - WebFormS, LDAPS, WebUploadS, RESTS, NNTPS, IPDaemonS, POPS, FileMailerS, SOAPS, HTMLMailerS, TelnetS, HTTPS, FTPS, IMAPS, SMTPS, IPPortS, CertMgr, RSSS, SMPPS, WebDavS, XMPPS

- **IP*Works! S/MIME**

Components for E-mail and file confidentiality, authentication, and non-repudiation through encryption and digital signatures. Implements the S/MIME standard for digital security.

Components - CertMgr, SNNTP, SPOP, SIMAP, SSMTP, SMIME, SFileMailer

- **IP*Works! SSH**

Secure Shell (SSH) enabled client communications components supporting strong encryption and advanced cryptography. A highly-evolved code base, enhanced with enterprise features like IPv6 addressing and 64-bit support.

Components - SFTP, SShell, SExec, SSHClient, SSHTunnel, CertMgr

COCOA FRAMEWORKS FOR INTERNET BUSINESS

Built using the same technologies as our award-winning IP*Works! product line, these packages offer native components for Credit Card Processing, Online Payments, E-Banking, Shipping & Tracking, and more!

- **QuickBooks Integrator**
- **E-Payment Integrator**
- **E-Banking Integrator**
- **TSYS Integrator**
- **Paymentech Integrator**
- **FDMS Integrator**
- **FedEx Integrator**
- **USP Integrator**
- **USPS Integrator**
- **PayPal Integrator**
- **Amazon Integrator**
- **SharePoint Integrator**

GET THEM ALL THROUGH OUR COMPREHENSIVE RED CARPET SUBSCRIPTION, $1,499 PER DEVELOPER, ROYALTY-FREE. 20% DISCOUNT FOR MACTECH SUBSCRIBERS.

Development in the Age of the **iPad**

Changes in the iPhone 3.2 SDK

by Rich Warren

Introducing the iPad

In a word, it's iTastic! Despite what seemed like a severe Internet backlash, the iPad has shown very strong initial sales, with Apple announcing that it sold one million iPads in the first 28 days. By comparison, the 1st generation iPhone took 102 days to reach the same one million mark. In fact, the initial demand was so great that Apple had to push back the iPad's international release by almost a month.

It's still too early to know if the iPad is really a new, disruptive technology; if it will truly be seen as a game-changing device. But, right now things look good for Apple, and what's good for Apple is good for Apple developers. Hoards of iPhone developers have eagerly jumped into the iPad app market. Initial evidence indicates that iPad apps sell at a higher average price point than their iPhone cousins. As an app consumer, that certainly seems to be the case. I'm sure I'm not the only person who bristled at paying $9.99 for an HD version of an iPhone app I already owned.

So, you might be asking yourself, "How do I catch the iPad gravy train?" Well, there's some good news and there's some bad news. First the good news, the iPad uses the iPhone SDK. That means, much of the knowledge and skills you learned developing iPhone applications will transfer over. However, from a user's perspective, the iPad has a very different feel than the iPhone. It's much more than a big iPod Touch. A successful iPad developer must understand a whole slew of new interface elements and idioms. Not to mention a host of new features added to the iPhone 3.2 SDK.

In this article, I will break these changes into three groups:

- Changes in Form and Use
- iPad-specific Interface Elements
- Other Additions to the SDK

The rest of this article will discuss each of these changes in more detail.

Changes in Form and Use

The most obvious change is, of course, the size. The iPad had a 1024 x 768 screen, compared to the iPhone's 320 x 480. This has two somewhat subtle side effects. First, the iPad's screen has a slightly lower resolution than the iPhone (132 ppi compared to the iPhone's 163 ppi). Also, the iPad's screen is more square (a 4:3 ratio, compared to the iPhone's 3:2).

The bottom line is, as a developer we have more space to play with. Of course, this means we must to do something intelligent with this space. We cannot just blow up an iPhone app and expect it to look nice. A table that looked great on the iPhone tends to look ridiculous when stretched across the whole iPad's screen.

On the other hand, iPhone's small screen forced developers to build tight, focused user interfaces (at least, it forced successful developers to build tight, focused interfaces). The iPad's extra space gives us the freedom to build richer applications—but it also gives us the ability to really clutter up the joint. There's a delicate balance here, and a lot of app developers are still struggling to get things right. A good user interface will make use of the extra elbowroom, while still maintaining the tight, focused feel of the iPhone.

And the usage differences are even more esoteric. For example, consider the average user's relationship with their apps. iPhone apps are typically developed for short interaction times. You're standing in line at the grocery store. You pull out your iPhone and waste a few minutes while you wait. The iPad, on the other hand, is more likely to be used on the couch or in a coffee shop. The user will spend more time with their apps at one sitting. This changes both the type of activities a user will want to perform, as well as the amount of work a user expects to accomplish. Oh, there will still be a need for quickie apps, but we no longer need to focus so heavily on the grocery-line use case.

There is also the issue of orientation. On the iPhone, it's OK to lock an application into either a portrait or landscape mode. Actually, I think this has changed somewhat over time.

For example, the original Mail app only worked in portrait mode, but it now supports landscape as well. However, for the iPad, Apple strongly recommends making all applications rotate freely into any orientation. And they mean ANY orientation; to the left, to the right, even upside down. Basically, you should be able to flip the device over as you hand it to someone, and the interface should rotate as needed.

Now, despite common wisdom, we don't always have to do everything Apple says. However, as a user, you come to expect certain things. It's easy to get into the habit of freely flipping the iPad around, and you quickly begin to notice (and become annoyed by) applications that don't rotate properly.

As a side topic, it's tempting to use orientation as a control. The user flips the iPad into landscape mode to activate one set of controls, flips it into portrait for another. Apple did this with their Pages app. In landscape, the app is full screen. Basically, all you can do is type. You must switch the iPad to portrait mode to access the formatting and document management controls.

While this can be an effective idiom for some applications, in general I think it's a mistake. First off, it makes the app harder to understand. iPad Apps typically don't have much in the way of documentation (and few people actually read what limited documentation there is). Users are often forced to experiment with a wide range of possible interactions (tap and hold, double tap, triple tap, two finger taps, three finger taps, etc.), just to see what the application can do. Changing the way the app behaves based on orientation only further obfuscates things. Also, consider the keyboard dock and Apple's iPad case. If you're using the keyboard dock, your iPad is forced into a portrait orientation. If you fold the case over to use it as a stand, you are locked into a landscape orientation. Either way, it feels odd to stop and flip the iPad around. You just want it to work the way it is.

Of course, it's important to remember that the iPad is still a very new device. No matter what anyone says, the actual idioms for use will grow as a collaboration between developers and users. In many ways, this is a brand new world. Developers will continue to show us new, exciting possibilities, while users select which ones they will actually adopt. For the foreseeable future, this will continue to be an experiment in progress.

iPad-specific Interface Elements

Navigation and tab bar views once ruled as kings of the iPhone developer's toolkit. They are still available on the iPad, but they have shifted to a largely secondary, support role. In the land of iPad, the split view now reigns supreme. The 3.2 SDK also made other interface changes to help utilize the iPad's larger screen real estate. In this section, we will cover the following:

- Split Views
- Popovers
- Modal View Presentation Styles
- Expanded Toolbars

Split Views

As the name suggests, the split view divides the iPad into two views…some of the time.

Here's how it works. When the iPad is in a landscape orientation, the split view displays two side-by-side views. The left view has a fixed width of 320 pixels. The right pane fills the rest of the screen (in this case, 704 x 768 pixels). When you switch the iPad to a portrait orientation, the left view disappears and the right view expands to fill the entire screen (now 768 x 1024 pixels wide).

The `UISplitViewController` class manages the side-by-side panes, including rotation and any other system-related behaviors. Meanwhile separate view controllers manage the actual content of the left and right views. The split view should always be the root view of an application's window. Specifically, you should never place a split view inside a navigation or tab bar. The contents of the left and right view can; however, be any arbitrary view.

Most often, the left view is used to display a list of items, while the right view displays details about the current selected item. This creates a slight problem when the iPad is in portrait mode, since the list view disappears, trapping the user in the most recently selected detail view. Typically, you work around this problem by adding a button to display the contents of the left view as a popover. The `UISplitViewControllerDelegate` supports this, through the `splitViewController:willHideViewController:withBarButtonItem:forPopoverController:` and `splitViewController:willShowViewController:invalidatingBarButtonItem:` methods.

An example implementation from Xcode's Split View-based Application template is shown below:

Code to setup and tear down the button and popover

The first method adds the provided button to the detail view's toolbar. It also assigns the popover to a property, thus retaining it. The second method removes the button and releases the popover.

```
// Called when the iPad is rotated into portrait mode,
// and the left view is hidden
- (void)splitViewController:(UISplitViewController*)svc
     willHideViewController:(UIViewController
*)aViewController
          withBarButtonItem:(UIBarButtonItem*)barButtonItem
       forPopoverController: (UIPopoverController*)pc {

    barButtonItem.title = @"Root List";
    NSMutableArray *items = [[toolbar items] mutableCopy];
    [items insertObject:barButtonItem atIndex:0];
    [toolbar setItems:items animated:YES];
    [items release];
    self.popoverController = pc;
}


// Called when the view is shown again in the split view,
// invalidating the button and popover controller.
- (void)splitViewController:(UISplitViewController*)svc
```

```
    willShowViewController:(UIViewController
*)aViewController
  invalidatingBarButtonItem:(UIBarButtonItem *)barButtonItem
{

    NSMutableArray *items = [[toolbar items] mutableCopy];
    [items removeObjectAtIndex:0];
    [toolbar setItems:items animated:YES];
    [items release];
    self.popoverController = nil;
}
```

Here, the split view creates the button and the popover controller for you. All you need to do is place the button on your user interface, and retain both the button and the popover controller. You then remove the button and release the objects when they are no longer needed. Note: in the sample code, the memory management is handled automatically by the `popoverController` property and the `NSMutableArray`.

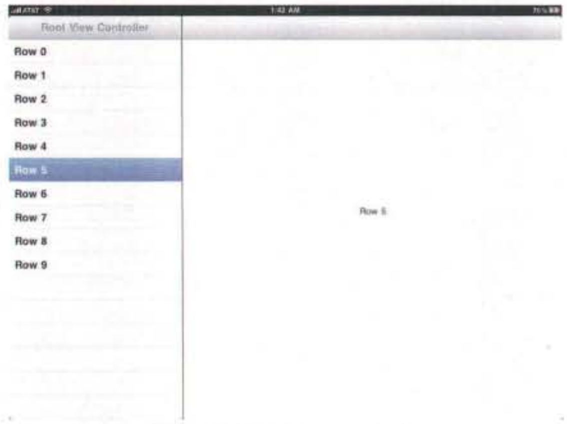


Figure 1: Split View Landscape

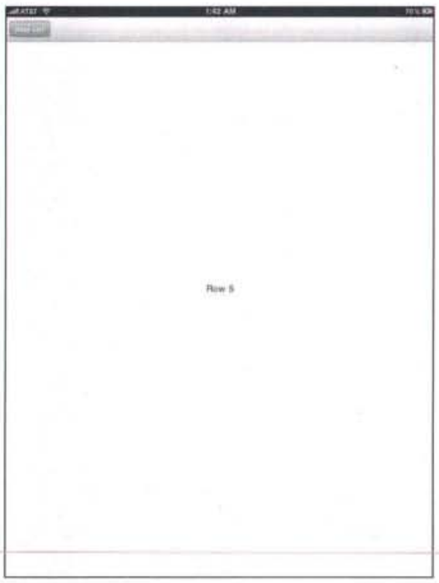


Figure 2: Split View Portrait

Note: Much like the tab view, the split view controller only supports autorotation if both the subviews support it. This is true even in portrait mode, where only the detail view is displayed. Therefore, you must override `shouldAutorotateToInterfaceOrientation:` in both subview controllers to return `YES` for all supported orientations.

Popovers

Popovers provide a lightweight method for laying information over top the main view. In many ways, they are similar to the iPhone's modal views, but they typically allow the user to continue performing actions outside the popover. Additionally, the popover is typically displayed next to an item that it should modify, and it typically has an arrow pointing to that object.

To display a popover, simply create an instance of the `UIPopoverController` class, and initialize it with the view controller you wish to display. Then call one of the two methods to present the popover.

`presentPopoverFromRect:inView:permittedArrowDirections:animated:` allows you to present a popup pointing to any arbitrary location in the screen. You can use the `permitttedArrowDirections:` parameter to force the popup into a particular orientation relative to the given rectangle; however, it is usually best to just pass `UIPopoverArrowDirectionAny` and let the popover decide its own placement.

Since popovers are often displayed in response to toolbar buttons, a specialized presentation method is provided: `presentPopoverFromBarButton-Item:permittedArrowDirections:animated:`. This will present the popover with an arrow pointing towards the specified button.

The popover's size can be controlled in one of three ways. First, you can set the content view controller's `contentSizeForViewInPopover` property. This defaults to 320 x 1100 pixels. You can further override this value by either setting the UIPopoverController's `popoverContentSize` property, or by calling its `setPopoverContentSize:animated:` method. In any case, the popover's width should be between 320 and 600 pixels. You can use any value for the height; however, the popover may be resized so that it fits on screen without covering the keyboard.

casper SUITE

WE HELP ENTERPRISE CUSTOMERS SUCCEED WITH THE APPLE PLATFORM

Learn about how the Casper Suite enables IT administrators to manage large-scale Mac deployments.

One to one Apple deployments with the Casper Suite

Video archive now available

Whether your school is getting ready for a new one to one deployment, is expecting a major hardware refresh or is simply facing another summer of imaging, the topics covered in this webinar can help you work effectively. Current Casper Suite administrators looking for best practice recommendations as well as other IT professionals looking for a better way to image Macs will find this presentation compelling.

View the archived webinar and read more about one to one deployments online:

http://www.jamfsoftware.com/solutions/one-to-one

Configuring the Mac OS for secure environments

New event announcement

Whether you are a Government, Commercial or Education organization, you will benefit from this webinar about using native tools within the Mac OS and the Casper Suite to configure your server, enforce security standards and create custom reports for demonstrating regulatory compliance.

Wednesday June 9th, 2010
2:00 pm CDT (GMT -5:00)

Register for this free webinar and read more about Mac OS security online:

http://www.jamfsoftware.com/solutions/security

Complete product and event information at **jamfsoftware.com**

JAMF software

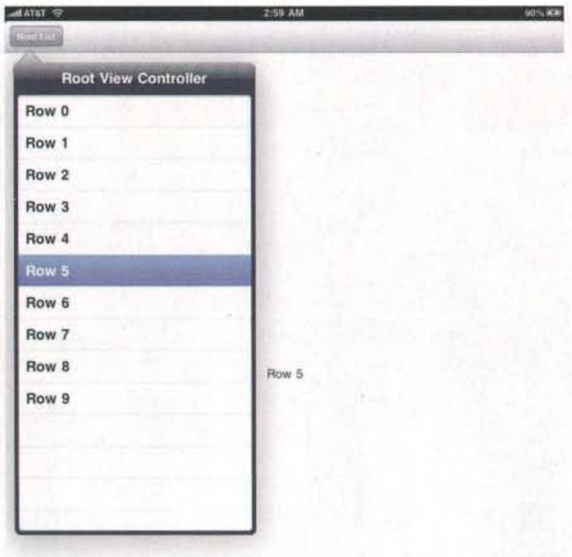


Figure 3: A Popover View Attached to a Bar Button

Modal View Presentation Styles

On the iPhone, whenever you present a modal view, it fills the entire screen. This has several advantages. It helps keep the iPhone interface simple and clean, while capturing the intent of a modal view. The user cannot do anything else until the view is dismissed (usually by making a selection within the view).

The iPad, however, gives us a little more control over how our modal views are displayed. Again, simply blowing up a view to fill the whole screen may not be the most visually appealing answer.

The `UIViewController` class now has a `modalPresentationStyle` property. It can be set to any of the following values:

`UIModalPresentationFullScreen`: the modal view fills the entire screen, taking into account the `wantsFullScreenLayout` property. If this value is true, the modal view will cover the status bar. Otherwise it will not.

`UIModalPresentationPageSheet`: The modal view's height is set to fill the screen, but the width is set to the screen's portrait width. In portrait mode, this is essentially the same as `UIModalPresentationFullScreen`. Any areas outside the modal view are dimmed to prevent the user from interacting with them.

`UIModalPresentationFormSheet`: The width and height of the modal view are smaller than the screen. The modal view is centered in the screen. If the keyboard is visible in landscape mode, the modal view is shifted upward as necessary.

`UIModalPresentationCurrentContext`: This view is presented using the same style as its parent view.

Other than these style values, modal views on the iPad and iPhone are exactly the same.

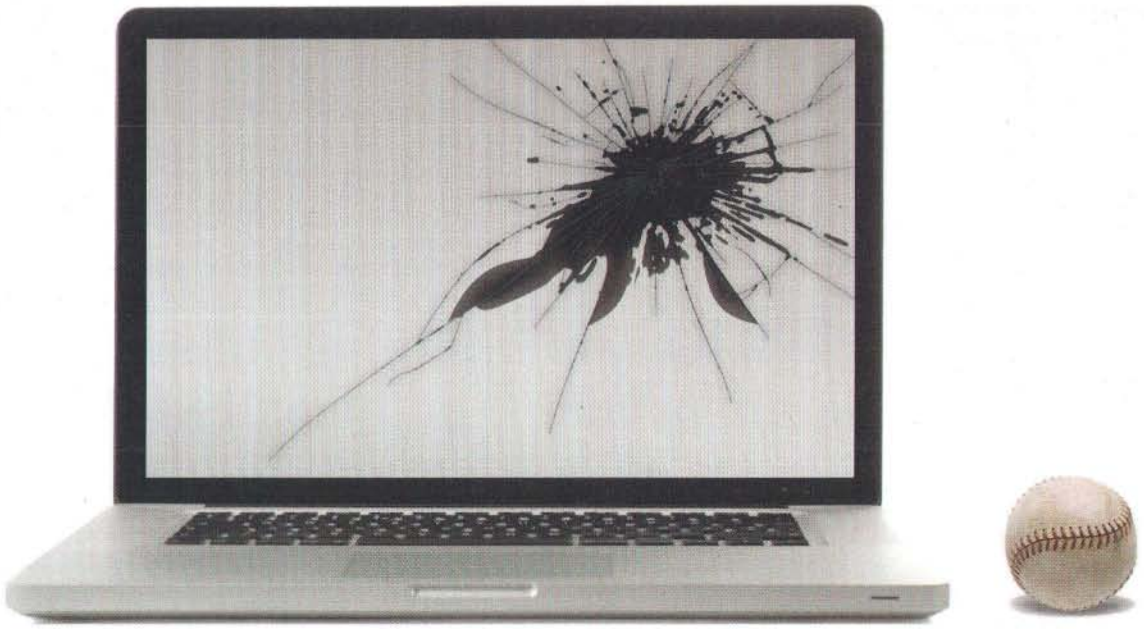
CONVERT DVI TO MINIDISPLAYPORT

Gefen introduces a new solution for enabling computers with DVI connectors to utilize new Apple displays using the MiniDisplayPort connection. The converter is a low cost solution available that makes the legacy computers useful for the foreseeable future.

Gefen

www.gefen.com

Stretch It. Switch It. Split It. Gefen's Got



WHEN LIFE THROWS YOU A CURVE BALL.

Clickfree's Transformer SE Backup Adapter converts any brand USB hard drive, iPod or iPhone into an Award-Winning, Clickfree Automatic Backup device.

Watch the demo video -

1 Plug it into your Computer with any external hard drive

2 Backup Starts Automatically - Your Files Are Safe!

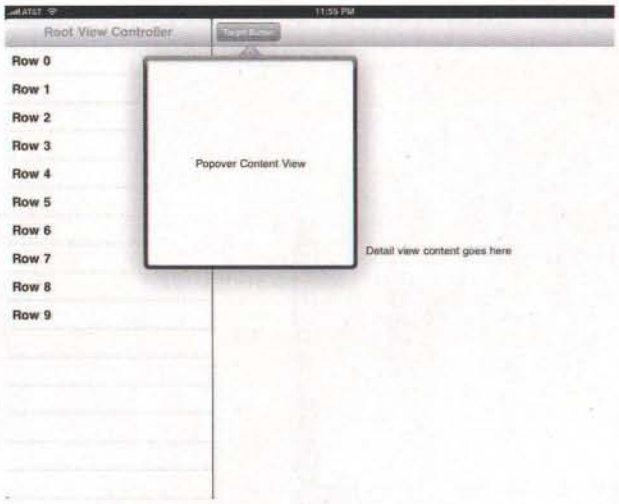


Figure 4: A Form Sheet Modal View

Expanded Toolbars

Finally, tool bars play a larger role in iPad applications. On the iPhone, the user interface guidelines recommended placing toolbars on the bottom of the screen, leaving the top for navigation bars. On the iPad, you can now place toolbars across the top of the screen instead. This makes the toolbar a more prominent part of the user interface. In addition, given the larger screen size, more items can be placed on the toolbar.

Other Additions to the SDK

The iPhone 3.2 SDK brought along a number of other additions. Unlike the interface changes discussed previously, most of these changes are not necessarily iPad specific, and some of these technologies should make it into the iPhone 4.0 SDK, allowing richer iPhone and iPod touch development as well.

To start with, we can create universal applications that will run on both the iPhone and iPad. From the user's perspective, this greatly simplifies the buying and managing of apps. You simply purchase one copy, and install it on all your devices. The application runs with a UI optimized for each device.

Developers can also create custom input views to replace the keyboard. This could, for example, allow the user to input math symbols or runic characters, or really anything you can think of. Additionally, you can attach an accessory view to the top of the system keyboard (or your own custom input view). This allows you to add additional controls to modify the input text. You have a lot of flexibility when creating your custom input views and accessories. The width must match the system keyboard, but you can use whatever height you wish. You can also place any number of sub-views and controls on the keyboard.

Additionally, you can give any `UIView` subclass (or any view class that inherits from `UIResponder`) the ability to receive text input. Simply implement the `UIKeyInput` protocol. Then, whenever the view is made first responder, the system keyboard will appear. You can further assign a custom

The Best Tensioned Screen Just Got Bigger

**Tensioned Cosmopolitan® Electrol®
Now Available Up To Sixteen Feet
SEAMLESS**

Weighted aluminum slat bar.

Two piece aluminum extruded case.

DA-LITE®

For a free catalog and screen recommendations, call toll free or visit us online.

1-800-622-3737

www.da-lite.com • info@da-lite.com
574-267-8101

input view or accessory by setting UIReponder's inputView or inputAccessoryView properties. Note: these properties are defined as read-only in UIResponder. You will need to override the properties and their backing variables. Then, instead of synthesizing the accessors, write a custom getter that loads or creates the input view as necessary.

The combination of arbitrary views receiving input from custom keyboard could result in very interesting (and in some cases, undoubtedly, very confusing) user interfaces. There's a lot of flexibility here. It will be interesting to see how developers use it.

Also, the 3.2 SDK provides support for gesture recognizers by using the UIGestureRecognizer class. These gesture recognizers can be attached to any view. The SDK provides recognizers for many common gestures: any number of taps, pinching, panning or dragging, swiping, rotating, and long touches (the so-called touch and hold). Additionally, you can develop your own custom recognizers.

Gesture recognizers are a large and complex topic. The API allows you to specify an order in which the recognizers are tested. This is important, for example, when trying to distinguish between a single and double tap. You must first look for double taps. If that fails, then look for a single taps. Otherwise, the single tap recognizer would consume all taps, and the double tap recognizer would never get called. You can even allow multiple recognizers to be triggered by the same gesture. Again, that's not something you'd typically want to do, but if the need arises, the possibility is there.

The iPhone 3.2 SDK also provides enhanced support for entering and displaying text, including Core Text for managing formatted text and complex layouts. It also adds Foundation-level regular expressions, a variety of tokenizers, spell check and word completion, and even an API for custom edit menu items (allowing you to go beyond the system defaults of Cut, Copy, Paste, Select, Select All and Delete). The text support is another broad topic, and it's worth examining in more detail than we can cover here.

Finally, the iPhone 3.2 SDK provides additional options for video playback, and an API for displaying content to an external display or projector. There is also a formal API for handling documents and files. You can register your application as capable of open specific file types. When you receive copies of those files (for example, downloaded through Safari or as a mail attachment), you're application will be added to the list of applications that can open that file. Similarly, when dealing with files of an unknown type, you can use a UIDocumentInteractionController object to handle the interaction—usually by either displaying a preview of the file's contents, or by providing a menu listing the applications that can open the file. You can also register your application to sync its files between the iPad and iTunes.

For more information on any of these topics, check out the iPad Programming Guide and the iPad Human Interface Guidelines. They can be found in the online reference library at http://developer.apple.com/iphone.

and you thought rabbits duplicated fast

866.4.RABBIT

- CD/DVD Manufacturing
- Content Creation
- Fulfillment / Distribution
- Marketing

www.amsrabbit.com

Advanced Security WORKSHOPS

Go4Cast, provides workshops on advanced security topics:

- Digital Signatures and Email Encryption
- Creating a Certificate Authority on OS X
- 802.1x Security on OS X SmartCards and 2-Factor Authentication

Both Classroom and Web Based Training Available

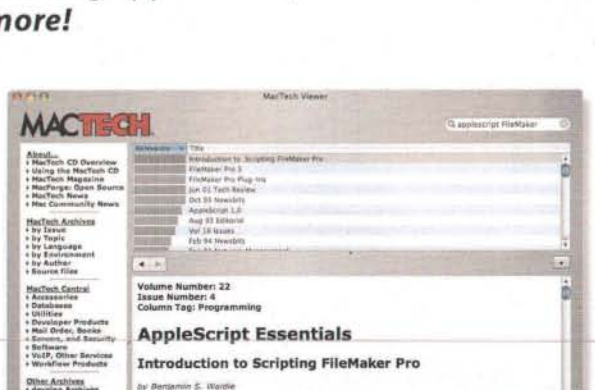
GO 4 CAST
ASSESS, ADAPT, ACHIEVE

8452 S. Federal Highway | Port Saint Lucie, FL 34952
888-247-1616 | go4cast.com

Join Us on Facebook!

The MacTech DVD - Volumes 1.01-25.12 is packed with more than ever before -- over 3200 articles from more than 290 issues **(1984 - 2009)** written by over 870 experts, all 29 issues of Apple's develop, 21 issues of FrameWorks magazine, 100+ MB of source code, MacTech Viewer, working applications, full documentation, demos for techs, ***and more!***

Everything is displayed in the very fast, very searchable ***MacTech Viewer!*** An application that has been designed specifically with Techs in mind. Search quickly through 25 years of great information provided by MacTech. Information to save you time, and make your life easier.



Volume Number: 22
Issue Number: 4
Column Tag: Programming

AppleScript Essentials

Introduction to Scripting FileMaker Pro

by Benjamin S. Waldie

For the past couple of columns, we have discussed various ways to store and access data using AppleScript. One column provided an introduction to Database Events, a background application in Mac OS X 10.4 and higher, which allows AppleScripts to interact directly with SQLite databases for the purposes of storing and accessing data. Another column explored methods of storing and accessing data in script properties and property list files. This month, we are going to continue discussing data storage and access, this time, using FileMaker Pro, a third-party commercial database application.

For the purposes of following along with this month's column, if you do not own FileMaker Pro, you can download a fully functional 30-day trial from **http://www.filemaker.com/**

All AppleScript code covered in this month's column was written and tested with FileMaker Pro version 8.0.1. Therefore, some of the AppleScript terminology discussed, may not function with

Mac

Universal

Requires Mac OS X v. 10.4.5 or later

Toll Free 877-MACTECH, Outside US/Canada: 805-494-9797 • http://www.mactech.com/dvd/

Conclusion

The iPad is a game changer for developers, if not for the world at large. The extra screen space lets us to do things that were simply not possible on the iPhone's smaller screen. More importantly, the changes brought with the iPhone 3.2 SDK help us take advantage of this real estate, while providing a set of interesting tools for breaking out of the iPhone box. There's a lot of power in the new APIs, and I can't wait to see what developers do with them.

The changes don't end here, though: Apple plans to release iPhone 4.0 for the iPhone and iPod touch this summer, with the iPad version coming later this fall. This will add a host of other improvements, including multitasking, Quick Look, video capture, iAd and the new Game Center.

The extended iPhone development environment is becoming a very interesting place indeed, and the iPad seems poised to become a key player in this space. MT

About The Author

Rich Warren lives in Honolulu, Hawaii with his wife, Mika, daughter, Haruko, and his son, Kai. He is an associate scientist with 21st Century Systems, Inc., a freelance writer and a part time graduate student. When not playing on the beach with his kids, he is probably writing, coding or doing research on his MacBook Pro. You can reach Rich at rikiwarren@mac.com, check out his blog at http//freelancemadscience.blogspot.com/ or follow him at http//twitter.com/rikiwarren

Why bore people with your iPhone pics...
when SonicPics creates audio-captioned slideshows on the spot?
• Photos with sound in sync
• Create custom slideshows
• Records up to 60 minutes (m4v)
• Share via email, WiFi, or YouTube
All you need is your iPhone!
SonicPics
www.sonicpics.com
Available on the App Store

Still using different tools for your cross-platform development?

Powerful source control where ever you work: Mac, Linux and Windows.

Get your **free** evaluation download at www.purecm.com/mactech.php

Cross-platform GUI and command line clients. Intuitive project history and easy to use visual tools for annotated file history or stream/folder compare.

Full support for task-based development. Private workspaces that allow you to define multiple changesets to reflect tasks. Easily checkpoint, rollback or merge changesets at any time.

Facilitate distributed and parallel development. Instant branching and visual merge support using a workspace or by merging directly on the server.

Impressive ROI. Better productivity thanks to full project transparency, support for agile development and reduced administrative overhead.

PureCM – software configuration management that grows with your needs. Now available as Standard and Professional edition.

PureCM purecm.com

by Dave Dribin

Calling All CPUs

Operation Cues and Grand Central Dispatch

Operation Queues

Back in the December 2009 article, we talked about the importance of keeping long running tasks off the main thread in order to keep the user interface responsive and avoid the SPOD or pinwheel cursor. In that same issue, we used threads and **NSThread** in order to move long running tasks off the main thread and on to a background thread. However, threads are a very low-level way to add concurrency to your application. It can be difficult to write multithreaded code safely. We talked about race conditions that can happen when two threads try and access the same variable at the same time and how to avoid them using the **@synchronized** keyword to lock critical sections. Other dangers of threads include deadlocks and live locks.

Mac OS X 10.5, Leopard, and iPhone OS 2.0 added the concept of operation queues to help with concurrency issues. To see how operation queues can help, let's first review the simple **NSThread** example from the December issue:

```
- (IBAction)runLongTaskWithThreads:(id)sender
{
    [NSThread
detachNewThreadSelector:@selector(longRunningTask)
                              toTarget:self
                            withObject:nil];
}

- (void)longRunningTask
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];
    NSLog(@"Starting task");
    [NSThread sleepForTimeInterval:10.0];
    NSLog(@"Task finished");
    [pool drain];
}
```

Here, we are creating a new thread in a user interface action to keep from blocking the main thread. Instead of just sleeping, let's do some real work. Say we have a text view in the user interface, and we want to count the number of words in it. This is normally a fast action and probably does not require threads, but let's say this is a *large* document and it could take several seconds to count all the words. This is not something we want to do on the main thread, so we could use a background thread like this:

```
- (IBAction)countWords:(id)sender;
{
    NSString * textViewString = [_textView string];
    [NSThread
detachNewThreadSelector:@selector(countWordsThread:)
                               toTarget:self
                             withObject:textViewString];
}

- (void)countWordsThread:(NSString *)string;
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc]
init];

    NSInteger numberOfWords = [self countWordsInString:string];
    NSLog(@"Number of words: %u", numberOfWords);

    [pool drain];
}
```

One thing worth pointing out here is that we are getting the text view's string on the main thread and then passing it to the background thread. Typically, when sharing data between threads, you must protect access to them with locks. However the contents of the **NSString** cannot change, because **NSString** is an *immutable object*. On the other hand, passing a non-thread safe object that can be changed, such as an **NSMutableString**, to a background thread must be carefully protected with locks to avoid race conditions. In general, it is a good idea to work with immutable objects and threads as much as possible, as it greatly simplifies the structure of your code.

While this use of threads is safe, and it moves the work to a background thread, one problem is that it always creates a new thread any time the user clicks on the "Count Words" button. If the user hits the button a lot, this can create a lot of threads in the system. If the number of threads exceeds the number of CPU cores available, it could still cause SPODs by starving the main thread of CPU it needs to run the user interface. There are techniques to limit the number of threads spawned using thread pools, but why bother when we have a better way?

Operation queues to the rescue. The first thing we need to do is break out the task of counting words into a separate work unit, called an *operation*. There's a class called **NSOperation** that represents this work unit, and one way to use it is to create a custom subclass that counts words:

Listing 1: WordCountOperation.h

```
#import <Cocoa/Cocoa.h>


@interface WordCountOperation : NSOperation
{
    NSString * _string;
}

- (id)initWithString:(NSString *)string;

- (void)main;

@end
```

The initializer takes a string to count words on, and the `main` method does the work of the actual word counting:

Listing 2: WordCountOperation.m

```
#import "WordCountOperation.h"


@implementation WordCountOperation

- (id)initWithString:(NSString *)string;
{
    self = [super init];
    if (self == nil)
        return nil;

    _string = string;

    return self;
}

- (NSInteger)countWordsInString;
{
    NSScanner * wordScanner =
        [NSScanner scannerWithString:_string];
    NSCharacterSet * whiteSpace =
        [NSCharacterSet whitespaceAndNewlineCharacterSet];

    NSInteger wordCount = 0;
    while ([wordScanner scanUpToCharactersFromSet:whiteSpace
                                       intoString:nil])
    {
        wordCount++;
    }

    return wordCount;
}

- (void)main;
{
    NSInteger numberOfWords = [self countWordsInString];
    NSLog(@"Number of words: %d", numberOfWords);
}

@end
```

As usual, I'm using garbage collection. If you're using manual memory management, don't forget to **retain** the string in the initializer and **release** it in a **dealloc**.

So, now that we have our word count task encapsulated into an **NSOperation** subclass, how do we use it? Enter the **NSOperationQueue** class. The **NSOperationQueue** class manages a queue of **NSOperation** objects. By default, the operation queue runs **NSOperations** added to the queue in background threads. The operation queue is smart, though, and doesn't try to create too many threads. The actual algorithm for how many threads it creates by default is implementation specific. What's important to know is that if you add more operations to the queue than the CPU cores can keep up with, it will wait until a previously submitted operation finishes before running a another one.

Here's what our **countWords:** action look like with the work of counting words delegated to an operation:

```
- (IBAction)countWords:(id)sender;
{
    NSString * textViewString = [_textView string];
    NSOperation * operation =
        [[WordCountOperation alloc]
            initWithString:textViewString];
    [_queue addOperation:operation];
}
```

The **_queue** instance variable is a previously created **NSOperationQueue** object. A good place to create this would be when the application finishes launching:

```
- (void)applicationDidFinishLaunching:
    (NSNotification *)notification
{
    _queue = [[NSOperationQueue alloc] init];
}
```

Instead of spawning a new thread, we add our new operation to an existing operation queue. Thus, with very little work, we are writing more efficient code than we would be with manual threads.

It's important to note that the **main** method does not get called on the main thread. We have to make sure we pass immutable objects to the operation, make our own copy of the object, or protect any mutable objects with proper locking. In this case, we're again utilizing the immutable nature of an **NSString** to simplify our threading uses.

Also, don't forget to set up an **NSAutoreleasePool** in the **main** method if you are using manual memory management. It's good practice to always setup autorelease pools in your **main** methods, the same way you create them for thread entry points.

Updating the User Interface from an Operation

In the code above, we are just logging the number of words to the console. This isn't that useful, and we will most likely want to update the user interface. For example, we might want to put the word count in a label. Remember that AppKit and UIKit user interface objects must *never* be accessed from a background thread, and this includes the **main** method of an operation. Thus, we need to pass the word count back to the main thread somehow. A good way to do this is to create a delegate for the operation subclass, and call the delegate method on the main thread.

The delegate would have a method that takes the number of words counted:

```
@protocol WordCountOperationDelegate <NSObject>

- (void)wordCount:(WordCountOperation *)operation
    didCountWords:(NSInteger)numberOfWords;

@end
```

We could have our application delegate implement this delegate as follows, to update the text of an **NSTextField** label:

```
- (void)wordCount:(WordCountOperation *)operation
    didCountWords:(NSInteger)numberOfWords;
{
    [_wordCountLabel setIntegerValue:numberOfWords];
}
```

On the **WordCountOperation** side, we need to update the initializer to take the delegate:

lowel EDU A LIGHTING RESOURCE CENTER
Lighting for HD Video
Adding Small Accent Lights (to Lead the Eye & Create a Sense of Depth)

Learn to light better.

Lowel EDU

Even with today's forgiving cameras and run-and-gun mentality, you still gotta light. To bring out the best in an interviewee, to add drama to product, separation to a background, and make your image stand apart from the competition. We've been making the world's best location lights for the motion picture industry for fifty years. And we have a free website designed for newcomers and old pros who want to brush up. Don't curse the darkness, go to Lowel EDU at Lowel.com.

lowel®
www.lowel.com

TIFFEN
The Difference is Tiffen

Learn to light better: Lowel EDU at www.lowel.com

```
- (id)initWithString:(NSString *)string

delegate:(id<WordCountOperationDelegate>)delegate;
{
    self = [super init];
    if (self == nil)
        return nil;

    _string = string;
    _delegate = delegate;

    return self;
}
```

In the `main` method we need to call the delegate method on the main thread. The trick is to use a `performSelectorOnMainThread:` call. The only problem with this method is that it can only be passed an object, not a primitive type like an `NSInteger`. This means we need to wrap up the primitive as an `NSNumber` and use an intermediate method to unwrap it on the main thread:

```
- (void)main;
{
    NSInteger numberOfWords = [self countWordsInString];

    NSNumber * wordCount =
        [NSNumber numberWithInteger:numberOfWords];
    [self
performSelectorOnMainThread:@selector(didCountWords:)
                              withObject:wordCount
                           waitUntilDone:NO];
}

- (void)didCountWords:(NSNumber *)wordCount;
{
    NSInteger numberOfWords =
        [wordCount integerValue];
    [_delegate wordCount:self
            didCountWords:numberOfWords];
}
```

This code counts the number of words in a background thread in `main`, and when that finishes, it calls `didCountWords:` on the main thread. This, in turn, calls the delegate, also on the main thread which allows the delegate to safely update the user interface without violating the AppKit threading constraints.

Other Ways to Create Operations

In the previous example, we created our own subclass of `NSOperation`, which nicely encapsulates the background task in a separate object. Sometimes this is appropriate, for example when we need to keep track of a lot of internal state for the background task. But sometimes it adds a lot of complexity for little benefit. In the previous example, we needed to create a delegate protocol and wrap up our results as an `NSNumber` in order to pass them to the main thread.

The first way to simplify this is to use an `NSInvocationOperation`, which allows us to use a target and selector as the entry point for an operation instead of creating a subclass:

```
- (IBAction)countWords:(id)sender;
{
    NSString * textViewString = [_textView string];
    NSOperation * operation =
        [[NSInvocationOperation alloc]
         initWithTarget:self
               selector:@selector(countWordsTask:)
                 object:textViewString];
    [operation autorelease];
    [_queue addOperation:operation];
}
```

When run on the operation queue, this **countWordsTask:** method runs on a background thread, just like the **main** method. This it also needs to pass the results to the main queue so it can update the user interface:

```
- (void)countWordsTask:(NSString *)string;
{
    NSInteger numberOfWords = [self
countWordsInString:string];
    NSNumber * wordCount =
        [NSNumber numberWithInteger:numberOfWords];
    [self
performSelectorOnMainThread:@selector(updateWordCountLabel:)
                             withObject:wordCount
                          waitUntilDone:NO];
}

- (void)updateWordCountLabel:(NSNumber *)wordCount
{
    NSInteger numberOfWords = [wordCount
unsignedIntegerValue];
    [_wordCountLabel setIntegerValue:numberOfWords];
}
```

We still have to use **performSelectorOnMainThread:** and wrap up the results as on object, but at least we don't have to create a subclass.

Believe it or not, it all gets even better. As of Mac OS X 10.6, Snow Leopard, **NSOperation** gained the ability to be created with a block. In fact, you can add a block directly to an operation queue. In order to help interact with the main thread, there is a special main operation queue that always runs operations on the main thread. Thus, we could implement our **countWords:** action with two nested blocks:

```
- (IBAction)countWords:(id)sender;
{
    NSString * textViewString = [_textView string];

    [_queue addOperationWithBlock:^{
        NSInteger numberOfWords =
            [self countWordsInString:textViewString];
        NSOperationQueue * mainQueue =
            [NSOperationQueue mainQueue];
        [mainQueue addOperationWithBlock:^{
            [_wordCountLabel setIntegerValue:numberOfWords];
        }];
    }];
}
```

This first block counts the number of words on a background thread. Then, we use the main queue to update the word count label on the main thread. This means we do not have to create a separate method just to use as an entry point, and we don't have to use a separate method for **performSelectorOnMainThread:**. Because blocks can capture local variables, we don't need to wrap up **numberOfWords** as an **NSNumber**, either. The second block on

STOP SHARING!

START FAXING!

Each subscriber receives faxes directly by email as PDF file attachments.

Corporate accounts from 3 to 100+ users available

For more information and a special offer for MacTech readers, visit

www.MaxEmail.com/MacTech

maxemail®

Call: 800-964-2793

the main queue can access the **NSInteger** directly, so this cleans up passing results to the main thread, too.

While using block and invocation operations may be useful in some cases, it's still useful to create **NSOperation** subclasses in many circumstances. If the background work is even mildly complicated, it's probably better to encapsulate this work in its own class. This provides a nice separation of concern and allows the operation to be reused as well.

The Maximum Concurrent Operation Count

As I mentioned earlier, an **NSOperationQueue** by default manages the number of threads running automatically. An operation queue does allow us to customize this, though using these two methods:

```
- (NSInteger)maxConcurrentOperationCount;
- (void)setMaxConcurrentOperationCount:(NSInteger)count;
```

The default value of the maximum concurrent operation count is a special value:

```
NSOperationQueueDefaultMaxConcurrentOperationCount
```

This value lets the system manage the number of threads to run at the same time. It's supposed to ensure that it never starves the CPU from other threads and will usually use the number of CPU cores as a guide. However, sometimes you need to tweak this, for example, if you use operations that perform a lot of I/O.

Sometimes you don't want to run the operations in parallel at all. In this case, you can set the max concurrent operation count to one. Then, only one operation will be run at a time. If any operation is currently running, then future operations get queued up. This kind of queue is called a *serial queue*, since all operations run one at a time, rather than in parallel.

The main queue *always* has a maximum concurrent operation count of one. This makes sense, as there's no way to run multiple operations simultaneously on the single main thread. Keep this in mind when you add an operation to the main queue. Also, remember that you should keep the work on the main thread to a minimum. Do as much work as possible on the background thread before running on the main queue.

Advanced Operation Usage

Setting the maximum operation count to one on your custom queues can ensure that only one operation at a time runs, as well. You may think this is useful if, for example, you wanted to ensure operations run in a particular order. Perhaps one operation needs the result of a previous operation. Tweaking the maximum operation count isn't a good way to do this, however. Operations can be made dependent on other operations. Take this snippet of code:

```
    // NSOperation * operation1 = ....
    [_queue addOperation:operation1];

    // NSOperation * operation2 = ....
    [operation2 addDependency:operation1];
    [_queue addOperation:operation2];
```

By adding **operation1** as a dependency to **operation2**, the operation queue will not start **operation2** until **operation1** is finished. This dependency mechanism is quite powerful. An operation can be dependent on multiple operations, and you can create a whole chain of dependent operations. The operation queue keeps track of all the bookkeeping to ensure they get run in the correct order.

Operations can also be cancelled. This allows the user to stop a long running task if the change their mind. Proper handling of cancellation is a bit complicated, and we don't have time to cover it in this article. However, many operations should properly support cancellation so that CPU utilization is not wasted and can be used for other purposes.

Grand Central Dispatch

Another feature added to Snow Leopard, named *Grand Central Dispatch*, often shortened to *GCD*, heavily uses blocks. GCD is a straight C library to help developers write concurrent code and take advantage of all CPU cores available on modern CPUs. While GCD does not require blocks, you will almost exclusively use the block-based GCD APIs because they are so much easier to work with than C function pointers. Unfortunately, blocks and GCD are not yet available on iPhone OS, so this section pertains to Mac OS only, at the moment.

In the simplest case, GCD has *dispatch queues* which are roughly similar to operation queues, except that they only work with blocks. In order to run blocks concurrently, you don't create your own dispatch queue. You use one of the global queues. Here's a snippet of code to run a block on a background thread:

```
    long priority = DISPATCH_QUEUE_PRIORITY_DEFAULT;
    dispatch_queue_t dispatchQueue =
        dispatch_get_global_queue(priority, 0);
    dispatch_async(dispatchQueue, ^{
        NSLog(@"On a background thread");
    });
```

Remember, this is a purely C-based API, so we're using straight C functions. The **dispatch_async()** function adds the block to the dispatch queue. The "async" part of the function name means it's asynchronous: it returns right away, and the block will run at some point in the future on a background thread. So this is very similar to adding an operation to an operation queue. In fact **NSOperationQueue** is built upon GCD in Mac OS X 10.6.

There is also a main dispatch queue available to run blocks on the main thread. This is considered a *serial queue* meaning the blocks submitted to it are never run in parallel. Again, this is similar to the main **NSOperationQueue**. Just like the main operation queue, it allows us to run blocks on the main thread to, say, update the user interface.

Here's how we would implement our **countWords:** action with GCD:

```
- (IBAction)countWords:(id)sender;
{
    NSString * textViewString = [_textView string];
    long priority = DISPATCH_QUEUE_PRIORITY_DEFAULT;
    dispatch_queue_t globalQueue =
        dispatch_get_global_queue(priority, 0);
    dispatch_async(globalQueue, ^{
        NSInteger numberOfWords =
            [self countWordsInString:textViewString];
        dispatch_queue_t mainQueue =
            dispatch_get_main_queue();
        dispatch_async(mainQueue, ^{
            [_wordCountLabel setIntegerValue:numberOfWords];
        });
    });
}
```

This looks very similar to the block operation variant. It really doesn't matter a whole lot if you choose operation queues or GCD, so choose whichever fits the situation best. The GCD approach is slightly simpler in this because we can use one of the global queues instead of creating our own operation queue. However, having our own operation queue can give us more control over which operations are added to it. Also, dispatch queues do not have the concepts of dependencies or cancellation. If you need these features, you are better off using an `NSOperation` subclass rather than using a dispatch queue.

More uses for GCD

So far, we've talked about two kinds of dispatch queues: the global concurrent queues and the main serial queue. We can also create our own custom dispatch queues using the `dispatch_queue_create()` function:

```
    dispatch_queue_t myQueue =
        dispatch_queue_create("org.dribin.dave.mactech",
NULL);
```

The first argument to this function is the name of the queue. While this is mainly used for debugging purposes, it is recommended to use a unique name. The best way to come up with a unique name is to use reverse-DNS notation, as I did above. The second parameter is reserved for future use and must always be `NULL` for now.

Blocks can be submitted to custom queues using the `dispatch_async()` function:

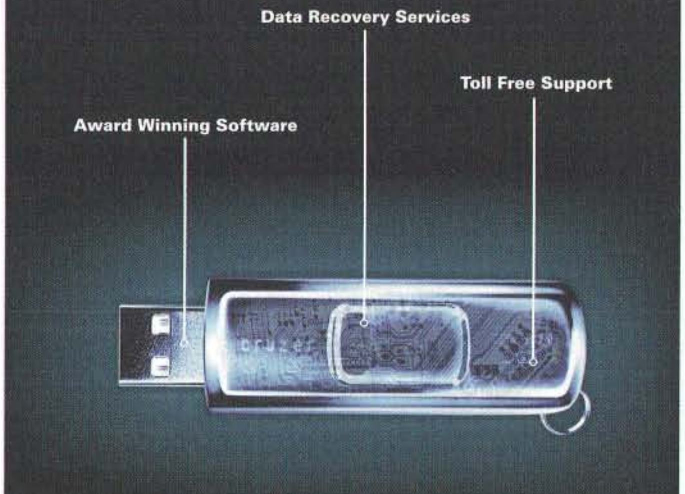
```
    dispatch_async(myQueue, ^{
        [self doSomething];
    });
```

While these blocks will run asynchronously in a background thread, custom dispatch queues are always serial queues. This means that they only run one block at a time, and all others are queued up. This is similar to an operation queue with a max concurrent operation count of one.

There is also a `dispatch_sync()` function which queues the block, but then waits for it to actually run before returning. With `dispatch_sync()` it is possible to use custom queues as a synchronization mechanism. The traditional locking used for getters and setters use the `@synchronized` keyword to protect critical sections:

```
- (void)setName:(NSString *)name
{
    @synchronized (self) {
        if (_name != name) {
            _name = [name copy];
```

RECOVER IN A FLASH



LC TECHNOLOGY INTERNATIONAL

www.LC-Tech.com/offers | (866) 603-2195

PDF Password Solutions
Save Filled PDF Forms
PDF Compress/Sign
PDF File Creation
Create PDF Forms
Process PDF Forms Data
Fill in PDF Forms Autom.

UNIVERSE SOFTWARE GMBH

www.pdf-office.com

extremely powerful and award-winning PDF software solutions

```
        }
    }
}

- (NSString *)name;
{
    NSString * name;
    @synchronized (self) {
        name = [[_name copy] autorelease];;
    }
    return name;
}
```

I'm showing the manual memory management here so you can see how it differs as we use GCD. This intrinsic lock ensures that the **_name** instance variable is correctly protected for use on multiple threads. Instead of using **@synchronized**, we can use a custom dispatch queue as follows:

```
- (void)setName:(NSString *)name
{
    dispatch_async(_dispatchQueue, ^{
        if (_name != name) {
            _name = [name copy];
        }
    });
}

- (NSString *)name;
{
    __block NSString * name;
    dispatch_sync(_dispatchQueue, ^{
        name = [_name copy];
    });
    return [name autorelease];
}
```

There are a couple of interesting points about this implementation. The setter doesn't look too different than it's locking variant, except we can use a **dispatch_async()**. This means that the setter returns immediately while the value is set concurrently in a background thread. This could be a performance benefit, and it's something that's not possible with traditional locks.

In the getter, we need to use a **dispatch_sync()** because we are returning a result. And as such, we need to mark the variable with the **__block** modifier, so it can be set inside the block. Also notice that the **copy** is done inside the block, while the **autorelease** is done outside. This is because the block may be run on a different thread, even though it's a synchronous call. As autorelease pools are thread specific, doing an **autorelease** inside the block would not carry through to the calling thread.

It is worth noting, though, that **dispatch_sync()** is not recursive, meaning you will deadlock if you try and nest multiple **dispatch_sync()** calls on the same queue:

```
dispatch_sync(_dispatchQueue, ^{
    dispatch_sync(_dispatchQueue, ^{
        // This is never actually run
    });
});
```

You do have to be a bit more careful than with intrinsic **@synchronized** locks, which are recursive. As with any synchronization mechanism, always do as little work as possible in the critical section. And be very careful about calling out to alien methods in the critical section, as that can lead to deadlock in unforeseen ways.

Conclusion

We looked at a couple of solutions that Apple has provided in order to help developers take advantage of multiple cores: operation queues and Grand Central Dispatch. Neither are magic bullets for concurrency, but both are very useful tools to have available in your tool belt to help you write safe, concurrent code. Believe it or not, this is just the tip of the iceberg for GCD. GCD has semaphores, timers, and can be used to monitor file descriptors. More information on these features will, alas, have to wait for a future article.

MT

About The Author

Dave Dribin has been writing professional software for over eleven years. After five years programming embedded C in the telecom industry and a brief stint riding the Internet bubble, he decided to venture out on his own. Since 2001, he has been providing independent consulting services, and in 2006, he founded Bit Maki, Inc. Find out more at <http://www.bitmaki.com/> and <http://www.dribin.org/dave/>.

An Awesome Email Marketing Tool for MACTECH READERS

Benchmark Email is the standard in permission-based email marketing

benchmark email
Email Marketing Services for your Business
Trusted by 65,000 Small Businesses & Users
FREE! 30 Day Trial

- ✓ **List Management**
- ✓ **Free Newsletter Archive**
- ✓ **Easy to Use Drag-n-Drop Email Editor**
- ✓ **Powerful Personalization**
- ✓ **Image Gallery**
- ✓ **Visual Graphs for Open & Clickthrough Tracking**
- ✓ **Spam & Spell Checkers**
- ✓ **Creative & Compelling HTML Templates**
- ✓ **Upload Your own Template**
- ✓ **Easy Video Integration**

MACTECH users get a 10% Lifetime Discount with this promo code 123923

Most email marketing services charge more money for less product. Not us. Benchmark Email's sophisticated suite of email marketing features lets you grow your list, send campaigns, track your data and even take online polls for an affordable price.

Plans starting at only $9.99 per month

Sign up for a FREE 30 Day Trial Today!

www.BenchmarkEmail.com

benchmark email

800.430.4095

Making PKG files with **The Luggage**

by Joe Block

Introduction

Bundling changes into PKG files is useful to a Macintosh administrator both for ease of creating master disk images and use by system management software. This article details a simple and more maintainable alternative to using Packagemaker's GUI for making OS X pkg files.

Why?

You've just been handed responsibility for maintaining a the master Macintosh disk image at your shop, and decided to use InstaDMG, which needs your additions to the stock image to be in Apple's PKG format. Or, maybe you're starting to use a system management tool like Puppet, Casper or ARD to manage your Mac herd and you need to bundle your updates and changes in Apple's PKG format to appease it. You start to create packages with Packagemaker and discover that it is tolerable, but has several notable usability flaws:

- It's not conducive to easily examining the differences between two versions of a given package making it difficult for coworkers to review your changes.
- You have to manually set up the directory tree you're going to package every time you make a new version of your package.
- You have to manually maintain the file ownership and permissions within the manually created directory tree.
- You have to remember to manually update the version number every time you build a new package.

I'm not fond of things that require manual setup steps - it's far too easy for things to become inconsistent between builds, which can lead to pernicious bugs that are hard to unravel.

Enter the Luggage.

While I was at Google, I wrote a tool that used GNU Make to allow my team to easily create packages by specifying their contents in a Makefile. This made detecting the changes between package versions easy - you could view the text diff in our internal code review system, which both made it easy to get a coworker to peer review your work and also easy to tell exactly what was supposed to be in the finished package without having to rummage through every screen and dialog in Packagemaker. Between generating the master disk images with InstaDMG and pushing software to our Macintoshes with Puppet, I made dozens of packages. Once I left Google, I wanted to have a similar tool available, so I wrote the Luggage, open sourced it and published it on github (http://github.com/unixorn/luggage).

What Luggage does for You

- Luggage enables quick and easy creation of packages that
- Install GUI applications that don't come with Apple PKG installers
- Wrap custom scripts & configuration files needed in your environment
- Generate payload-less packages that perform configuration changes in their postflight scripts. These can be pushed to your machines with a method of your choosing (Puppet/Casper/ARD/Custom script/etc.).

Getting Started

You'll need to have Apple's Developer tools installed on your build machine. You'll also need to have git installed (use a package manager like MacPorts or Fink, or, grab the Mac OS X git binary installer from http://code.google.com/p/git-osx-installer/). Git lets you get the current version of the source from github. Download the Luggage source by typing `git clone git://github.com/unixorn/luggage.git`, then cd into the Luggage directory and type `make` bootstrap_files to copy the Luggage's support files into /usr/local/share/luggage.

How it Works

The Luggage is a set of rules for GNU Make that describe how to set up the package root, copy your local files into it, set the correct ownership and permissions for the files, generate a pkg file and finally wrap it into a dmg file. By setting a few variables in your Makefile to configure a few parameters and list the payload of files you want in your finished pkg, you can then let Luggage handle all the tedious details of package building.

Your First Package

Package makefiles must be named Makefile (case matters here), include luggage.make and at a minimum, set the following variables:

- TITLE - the title of your package - It can't have spaces or punctuation since it's going to be part of the package's unique id.
- REVERSE_DOMAIN - your domain in reverse order (com.example.corp, for example), used along with TITLE by Luggage to create a unique identifier for your package.
- PAYLOAD - the list of files to be installed by the generated package.

Here's a small sample - note that luggage.make must be included before any variables are set so your variable definitions override the default values included in luggage.make.

```
include /usr/local/share/luggage/luggage.make
TITLE=install_foo
REVERSE_DOMAIN=com.example.corp
PAYLOAD=
```

Let's re-package a GUI Application

Several applications we want on our Macs at work don't come packaged in PKG format. This is annoying, both when I'm creating a master image with InstaDMG and when I'm pushing an update out with our system management software, puppet. So for my first example, we're going to repackage Firefox since it didn't come with a PKG installer. First, we create a tarball of the application (make sure to use Apple's tar so resource forks don't get mangled) with /usr/bin/tar cvf Firefox.app.tar Firefox.app, then we compress the tarball with bzip2 -9 Firefox.app.tar, and finally we update PAYLOAD in the Makefile with

```
PAYLOAD=unbz2-applications-Firefox.app
```

Our Makefile should now look like:

```
include /usr/local/share/luggage/luggage.make
TITLE=firefox
REVERSE_DOMAIN=com.example.corp
PAYLOAD=unbz2-applications-Firefox.app
```

This tells luggage.make that we want Firefox.app.tar.bz2 untarred into /Applications in our package. Now we can create a package with "make pkg", or have the package automatically wrapped in a disk image with "make dmg". That's all it takes. By default, the version number of the package is the current date in YYYYMMDD format, but you can set it manually by setting PACKAGE_VERSION=SOMETHING_THAT_MUST_BE_NUMERIC.

Packaging scripts and configuration files

I have several site-specific scripts that I install on my Mac herd in /usr/local/bin. I also want to preconfigure Kerberos and LDAP. I wanted it to be easy to package them for initial image creation with InstaDMG.

For each destination the Luggage supports, there is a pack-path-to-destination-split-by-dashes-filename rule. Each of the pack rules work on the assumption that whatever filename you have in the working directory is the same name you want installed in the destination directory by the package. If you have a script that you want installed as /usr/local/sbin/check_on_corporate_network, you need to name it check_on_corporate_network, and then add a target pack-usr-local-sbin-check_on_corporate_network to PAYLOAD. PAYLOAD can have as many targets as you like, separated by spaces. As in shell scripts, you can put a "\" on the end of the line to show that the next line is a continuation. See Table 1 for a list of current built-in PAYLOAD types.

Here's a slightly more complex example that includes a custom Make stanza in addition to the stock Luggage rules - this is based on the Makefile I use to create the package that preconfigures LDAP on newly imaged machines.

```
include /usr/local/share/luggage/luggage.make
TITLE=configure_ldap
REVERSE_DOMAIN=com.example.corp
PAYLOAD=pack-ldap.conf \
    pack-directory-service-preference-
ContactsNodeConfig.plist \
    pack-directory-service-preference-
DSLDAPv3PlugInConfig.plist \
    pack-directory-service-preference-
DSRecordTypeRestrictions.plist \
    pack-directory-service-preference-
DirectoryService.plist \
    pack-directory-service-preference-
SearchNodeConfig.plist

# here's an example of creating a custom install for a
file
# destination that is unsupported by the Luggage.
# We're taking advantage of the existing Luggage support
for
# creating /etc by having our l_etc_openldap rule depend
on l_etc

l_etc_openldap: l_etc
    @sudo mkdir -p ${WORK_D}/etc/openldap
```

rule name	installed ownership	installed permissions	destination
pack-etc-foo	root:wheel	644	/etc/foo
pack-usr-bin-foo	root:wheel	755	/usr/bin/foo
pack-usr-sbin-foo	root:wheel	755	/usr/sbin/foo
pack-usr-local-bin-foo	root:wheel	755	/usr/local/bin/foo
pack-usr-local-sbin-foo	root:wheel	755	/usr/local/sbin/foo
pack-hookscript-foo	root:wheel	755	/etc/hooks/foo
pack-Library-LaunchAgents-foo	root:wheel	644	/Library/LaunchAgents/foo
pack-Library-LaunchDaemons-foo	root:wheel	644	/Library/LaunchDaemons/foo
pack-Library-Preferences-foo	root:admin	644	/Library/Preferences/foo
pack-ppd-foo	root:admin	644	/Library/Printers/PPDs/Contents/Resources/foo
pack-user-template-plist-foo	root:wheel	644	/System/Library/User\ Template/English.lproj/Library/Preferences/foo
unbz2-applications-foo	root:admin	based on tarball contents	/Applications/Foo
ungz-applications-foo	root:admin	based on tarball contents	Applications/Foo
unbz2-utilities-foo	root:admin	based on tarball contents	/Applications/Utilities/Foo
ungz-utilities-foo	root:admin	based on tarball contents	/Applications/Utilities/Foo

Table 1: Currently available PAYLOAD entry types

```
    @sudo chmod 755 ${WORK_D}/etc/openldap
    @sudo chown root:wheel ${WORK_D}/etc/openldap

# and now that we've defined l_etc_openldap, we can add
it as
# a dependency to the Makefile stanza that actually
installs ldap.conf.

pack-ldap.conf: l_etc_openldap
    @sudo ${INSTALL} -m 644 -o root -g wheel ldap.conf
${WORK_D}/etc/openldap
```

Check the source code for updates to this list of payload rules.

Adding Pre/Postflight and Other Scripts

You can also include InstallationCheck, VolumeCheck, preflight, postflight, postupgrade, and postinstall scripts in Luggage generated packages. If you name a script "preflight", "postflight", "postinstall", or "postupdate", you can then add pack-script-XX to PAYLOAD to have it automatically be added to the final package and run at the appropriate point in the package install process (where XX is one of "preflight", "postflight", "postinstall", or "postupdate").

Useful Make targets

make dmg – create a package, then wrap it in a dmg. The resulting dmg file will be put in the current directory.

make pkg – create a package and copy it into the current directory

make pkgls – create a package, then list the contents so you can confirm it's generating a package with a payload that matches what you're expecting.

Customizing your packages

Some of the assumptions made in the Luggage may not suit your environment, particularly about file ownership or permissions. While you could write a postflight script to make the changes, for your convenience, luggage.make runs `make modify_packageroot` after it has created the package root and copied the all the files listed in `PAYLOAD` there, but before it invokes Packagemaker's command line tool to create the package.

If you add that target to your package's Makefile, it will override the dummy one in luggage.make and let you change do whatever you want to your package root. When writing your make rule, remember that `${WORK_D}` contains the path to your package root. For example, if your package installed a modified `cupsd.conf` to `/etc/cups`, and you needed it to have a group of _lp, you would add the following to your package Makefile:

```
modify_packageroot:
```

```
@sudo chgrp _lp ${WORK_D}/etc/cups/cupsd.conf
```

If automatically setting the package version to the date you built the package isn't acceptable to you, you can set it to something different than YYYYMMDD. If you set PACKAGE_VERSION=must_be_numeric in your Makefile, the Luggage will use that instead. Similarly, you can set the PACKAGE_MAJOR_VERSION and PACKAGE_MINOR_VERSION variables, but the values must also be numeric.

Conclusion

I hope this introduction has been helpful, and that you can use The Luggage in your environment. The source on github includes several example Makefiles to get you started, and I'm open to suggestions for more features. Patches are welcomed.

MT

About The Author

Joe has been a Macintosh and Unix system administrator for many years in environments ranging from higher education to companies varying in size from 5 to 20,000 employees. He is currently a system administrator at Ooyala in Mountain View, CA. When he isn't working with computers, he is an avid photographer and foodie and can usually be found exploring for photography opportunities or new restaurants to try.

GraphicConverter 6

Save 10%

The universal genius for picture editing

- More than 1.5 million users
- Import of more than 200 graphic formats
- Export of more than 80 graphic formats
- Picture editing
- Document browser
- Slide show and batch processing
- Editing of all meta data (EXIF, IPTC, XMP, ...)
- And much more ...

Only $34.95
(Version in the box $44.95)

Save 10% by ordering direct from:
www.lemkesoft.com/mactech

Download the trial version

MacResource
Computers & Service
Your Resource for All things Mac

Parts, Parts, Parts
We have parts for iMac, G4/G5 towers, LCD panels G4/G5/Intel iMacs, MacBooks/Pros and Displays.

Logic Boards
G5/Intel iMac: from $179/$399
G5/Intel Towers: $299/$699 & up
G4/G5/Intel Xserve: $149/$299/$899
MacBooks/Pros Call for Pricing

Power Supplies
G4 iMac 15/17/20" $49/$79/$99
G5 iMac 17/20": $99/$129
G5 Towers: $169/$199
G4/G5/Intel Xserves $169/$219/$299
Xserve Raids $239.99

Xserve Processors
G4 1.33GHz DP: $99
G5 2.0/2.3GHz: $149/499
Intel 2.0GHz C2D $699
Logic BD/PS require exchange

AirPort Cards
Standard/Extreme: $69.99

Processors
For, MacPros, G5s, Xserves
Dual Processors (per module):
1.8/2.0/2.3GHz: $199/249/349
2.5DP/QP/2.7DP: $299/599
G5 1.6/1.8GHz: $199/299

Tower Systems
G5 1.6/1.8GHz $429/479
G5 1.8/2.0GHz DP $549/599
G5 2.3/2.5/2.7GHz DP $699/799/899
G5 2.5GHz Quad Dual DVI $999
Intel 2.66/3.06 GHz, MacPro $1499/1799

Intel iMacs!!!
1.83/2.0 GHZ 17" $549/599
2.0/2.16 GHZ 20" $629/649
2.16/2.33 GHZ 24" $729/749

Aluminum iMacs
2.0/2.4/2.66 GHZ 20" Snw Lprd $799/$899
This units are in Factory Refurb Boxes but Have minor scratches on the cases & 90 day warranty.

Need MacBooks/MacBook Pros?
MacBook2.0/2.16 C2D 13" $649/699
MacBook Pro 1.83/2.0 CD 15" $599/799
MacBook Pro 2.4/2.5 C2D 15" $849/899
MacBook Pro2.1/2.4 C2D 17" $949/1199

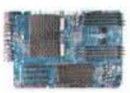
1-888-Mac-Resource
www.mac-resource.com

We Have G4/G5 & Intel Xserves & RAIDs Even if Apple Doesn't!!!!
G4 Xserve Full Unit: $299
G5 Xserve Cluster Node: $499
G5 Xserve Full Unit: $999
Intel Xserve 2.0ghz Full Unit: $1699
1TB Xserve RAID from $1999
2.8TB Xserve RAID from $2499
5.6TB Xserve RAID from $3699
3.5/7.0TB Xserve RAID: $2899/4299
Blank SATA/PATA Module $199
400GB SATA/PATA Module $399/$349
We carry Fibre Cards. Procs, PSs
Overnight Service Available

Refurbished Displays
Aluminum
20/23 Cinema(DVI): $299/599
30 Cinema(DVI): $999

Crystal
22/23" Cinema(ADC): $249/299
15" Studio LCD(ADC): $49
17" Studio LCD(ADC): $99
17" CRT, ADC/VGA: $49.99

Mac Minis GALORE, GREAT WORKSTATIONS!
1.42G4 GHz/256MB/40GB/Combo: $299
1.66/1.83/2.0 GHZ Intel Minis $349-$449
All Products are refurbished or demo call for more information.



Source Metrics on **Xcode**

Writing metrics for Xcode's new script system

by José R.C. Cruz

Introduction

To best manage a software project effectively, the project leader needs a way to quantify the project. She (or he) may need to monitor the output of each team member with minimal disruption. Managing a project is a challenging task, especially when the project itself is constantly changing. Some parts of the project may be incomplete, let alone stable enough for usability testing. Other parts may prove unwieldy, requiring change or removal. One way to quantify a given project is with the use of *source metrics*.

In this article, we look at two types of metrics that we can implement in our Xcode setup. First, we study the importance of such metrics, as well as their pertinent issues. Then we examine the new script system that came with Xcode 3.0. And we learn how to implement the source metrics as menu scripts using Python.

All featured menu scripts assume ANSI C as the source language. Readers are welcome to download the scripts and modify them for their own use. To get a copy of the menu scripts, go to the MacTech ftp site at ftp.mactech.com/src/mactech/volume26_2010/26.05.sit

Readers are expected to know their way around Xcode and to have a basic knowledge of Python.

The Concept of Source Metrics

Source metrics, or *software metrics*, is a family of algorithms that analyze various aspects of a software project or its source files. They work by breaking down the source into its constituent parts, counting those parts and using the results in an empirical formula. Source metrics are *static* metrics—they do not require sources to be compiled first. They are also hardware agnostic and often language dependent.

Uses of metrics

Source metrics offer many insights into the underlying structure of each source file. If properly used, they can aid us manage our projects and prepare our test plans. With source metrics, we can set practical goals for building or maintaining a given project. We can track the progress made to each source file and mark those files that may be error prone. Plus, we can better direct developer focus to those affected files.

We can also use source metrics to seek out potential trouble spots. We can identify those parts of the project that are subject to changes in user needs. Source metrics can even aid us in creating effective routines with which to test those parts.

Limits of metrics

On the other hand, each source metric examines only a *specific* aspect of a given project. Some metrics track only its code size, others its code complexity. So separately, source metrics are inconclusive—their results must be analyzed as a *whole* and in conjunction with *human* feedback.

Also, some source metrics are poorly defined and can give imprecise results. Some may even give varying results for the same project or source. To avoid this, only use source metrics known to be *reliable* and *valid*. These metrics give results that are both consistent and stable. They rely on actual content and can predict specific outcomes. It also helps to test the metrics against a *complete* project or source with *known* qualities.

Finally, source metrics are prone to abuse, often leading to projects being micromanaged. Misuse of metrics can cause dissent amongst team members, impede productivity, and erode morale. It is essential to balance metrics with real human factors like team input, code reviews and usability tests.

The Xcode Script Menu, Revisited

With version 3.0 of Xcode comes a new script system with which to control and enhance a development session. As in earlier versions, Xcode lists its scripts in its own Script menu (Figure 1). It supports most script languages, which includes Perl, Python and good old bash. It also has limited support for AppleScript via the command-line tool `osascript`. Choosing a script from the menu runs the script, its input being either the active source document or the entire project. Output from the script can appear in a dialog window, inserted in the source document, or saved to an external file.

Xcode can group its scripts based on common function. Each group then appears as a hierarchical menu list on the Script menu. Xcode can even assign each script with a unique shortcut key. This allows the use of a key-combo to invoke each script.

Changes to the menu

There are several changes to the new Xcode script system. First, Xcode did away with its `Scripts` directory. Instead, it keeps all its scripts in one XML file named `XCUserScripts.plist`. This allows Xcode to update its Script menu faster and more reliably. Second, Xcode comes with its own script editor. With this editor, we can add or change a script in the Script menu, create a new script group, assign shortcut keys, and control how each script interacts with Xcode.

On the downside, Xcode no longer allows the use of a third-party editor (like BBEdit) to add or edit one of its scripts. Also, it is no longer able to create and install its own menu script.

Managing the menu

At the end of the Script menu is the menu item Edit User Scripts. Choosing this item displays the editor window in which we manage the scripts that appear on the menu (Figure 2). You do not need an active project or source window in order to access this window.

The window itself has two distinct panes. The left pane (Figure 3) is where we control how each script appears on the menu. It is also where we collect related scripts under a single group. The pane presents the scripts as a hierarchical list. The left column of the list displays the *name* of the script or group, the right column the script's *shortcut key*.

Near the bottom of this pane is a button toolbar. The '+' button (Figure 4) lists the actions used to manage the list. Choosing New Shell Script adds an empty placeholder script named `Shell Script`. Xcode inserts the empty script *after* the last selected script. If the selection is on a script group, Xcode inserts the empty script at the *head* of that group. Choosing New Submenu adds an empty script group, named `Group`, to the menu list. Xcode also places the new group appears after the selected script. If selection is another group, Xcode places the new group at the head of the enclosing group.

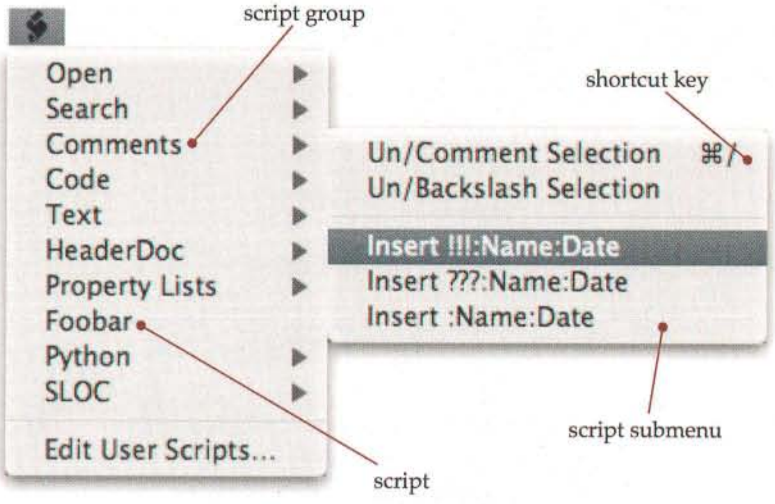
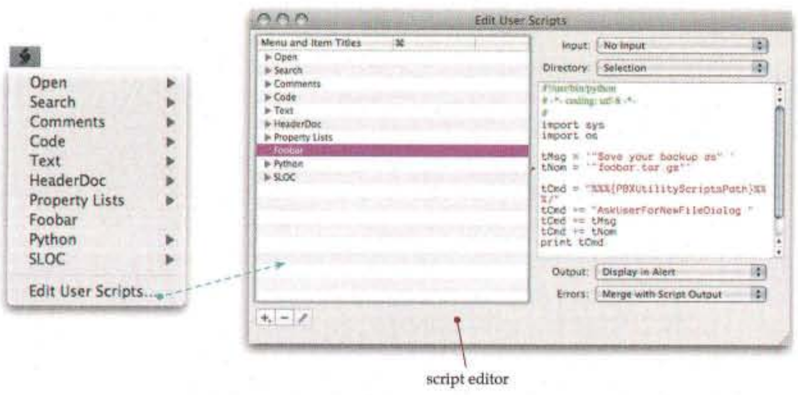


Figure 1. The Xcode script menu.



Input: No Input
Directory: Selection

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
import sys
import os

tMsg = '"Save your backup as" '
tNom = '"foobar.tar.gz"'

tCmd = "%%%{PBXUtilityScriptsPath}%%%/"
tCmd += "AskUserForNewFileDialog "
tCmd += tMsg
tCmd += tNom
print tCmd
```

Output: Display in Alert
Errors: Merge with Script Output

Figure 2. The script editor window.

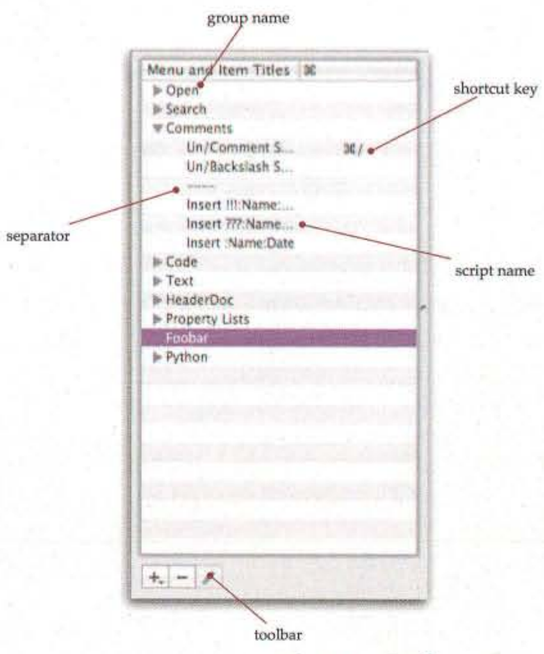


Figure 3. The list pane.

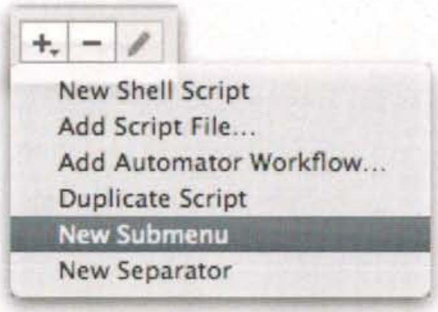


Figure 4. List of toolbar actions.

Choosing New Separator places a *dividing line* after the recent selection. Choosing Add Script File... displays an Open File dialog with which we can load an *external script file*. Choosing Add Automator Workflow... does the same thing, except we get to load a *workflow file*. And choosing Duplicate Script creates a *copy* of the script selected from the menu list.

The '-' button removes a selected item from the menu list. If the item is a script, Xcode removes the script from its `XCUserScripts.plist` file. If the item is a group, Xcode removes the group *and* its load of scripts. Removal occurs *immediately*, without any warnings. It is also *undoable*.

Renaming a script or group is a simple process. Just click its name on the list and type in the new name. Then press `Enter` to commit the name. Assigning a shortcut key is also quite simple. Just click the space next to that name and under the Command column. Then type the *letter key* you want to use as the shortcut. Make sure the assigned shortcut does not match any of the shortcuts used by other scripts or by Xcode itself.

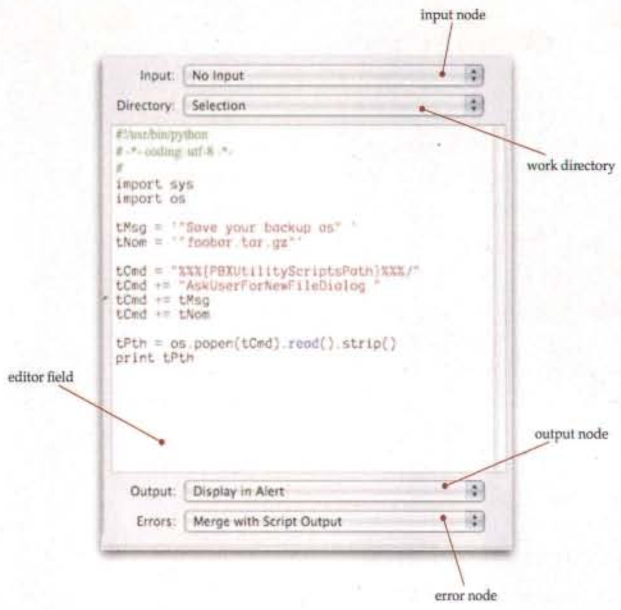
Making a script

The right pane of the editor window (Figure 5) is where we define the menu script. The script code goes into the *editor field*, which takes up most of the pane. Atop that field are two pop-up widget—they control the flow of input to the script. The Input widget sets input sources. To use the entire source document, click the widget and choose Entire Document. To use only a selected portion of the document, choose Selection from the pop-up list. In short, the Input widget defines the `stdin` node of the given script.

The Directory widget sets the work directory for the menu script. To use the same location as the active source file, click on this widget and choose Selection from the list. To use your home directory `'~'`, choose the list item Home Directory. To use the root node `'/'`, choose File System Root.

Below the editor field are two more pop-up widgets. They control the flow of output from the menu script. The Output widget sets the `stdout` for the given script. To write the output onto the active source document, click the widget and choose Replace Selection from its list. To display the output onto a dialog window, choose the list item Display in Alert. To dump the output to `/dev/null`, choose Discard Output.

The Errors widget sets the `stderr` for that same script. To display the errors on a dialog window, click this widget and choose Display in Alert. To route the errors to `stdout`, choose the list item Merge with Output. And to dump the errors, choose Ignore Errors.



Input: No Input
Directory: Selection

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
#
import sys
import os

tMsg = '"Save your backup as" '
tNom = '"foobar.tar.gz"'

tCmd = "%%%{PBXUtilityScriptsPath}%%%/"
tCmd += "AskUserForNewFileDialog "
tCmd += tMsg
tCmd += tNom

tPth = os.popen(tCmd).read().strip()
print tPth
```

Output: Display in Alert
Errors: Merge with Script Output

Figure 5. The editor pane.

Counting the Lines

The most common, and perhaps the most abused, source metric is **SLOC** (*source lines of code*). SLOC has its origins in the early years of digital computing. During that time, a punch card is exactly one line of executable or non-executable code. Thus, SLOC estimates the size of a given project or source by the *number of code lines* it contains.

SLOC is a simple metric, quite easy to implement and analyze. It gives non-technical people vital clues to the size and "complexity" of a given project. Also, some source metrics use SLOC as one of their primary input data.

Physical SLOC

There are several types of SLOC metrics. The simplest type, *physical SLOC* (Listing 1), counts the number of lines in a source file. This type does not differentiate between executable or non-executable code. Nor does it check if each line is part of a loop or a branch.

Physical SLOC uses a file's native `EOL` token to mark the end of each line. For Xcode source files, that token is a line feed (`0x0a`). For MPW files (Macintosh Programmer's Workshop, to you young blokes), that token would be a audioengine **upgrade your music!**

GET 10% OFF plus free shipping when you buy online! Use Coupon Code **MTTEN**

Experience our award-winning sound, high-quality materials and truly useful features.

$199 per pair

Audioengine 2 (A2)
Premium Powered Desktop Speakers

Closes the gap between computer speakers and home audio

*"These **are** the best speakers for your desktop, computer, or media player."* – Connect Reviews

Made for iPod

$119 per set

Audioengine W2 (AW2)
Premium Wireless Adapter for iPod

Unwire your iPod

"Super fast setup and the uncompressed sound is pretty remarkable." – Uncrate

Model A5N shown in Solid Carbonized Bamboo $449

starting at **$349** per pair

Audioengine 5 (A5)
Premium Powered Bookshelf Speakers

High-quality audio for your Mac or PC

"There are no other speakers in this price range that come close." – Mac Observer

$99 per set

Audioengine W1 (AW1)
Premium Wireless Audio Adapter

Send CD-quality wireless anywhere with or without a computer

"High quality wireless music streaming that's quick and easy to use." – Register Hardware

Works with all your gear • All cables included • 30-day audition • 3 year warranty

Visit our website for more product info, reviews, and awards: **www.audioengineusa.com** carriage return (`0x0d`). But physical SLOC ignores tokens that mark *separate* executable lines. Thus, the following code snippet will have a physical SLOC of *three*.

```
    c = 0;
    for (i = 0; i <= 100; i++)
        c++;
```

But this snippet has a physical SLOC of *one*.

```
    c = 0; for (i = 0; i <= 100; i++) c++;
```

Listing 1. Counting physical lines of code.

```
#!/usr/bin/python
# import the following modules
import sys
import re

# read the data in STDIN
tInp = sys.stdin.readlines()

# count the number of physical code lines
tCnt = 0
for tLne in tInp:
    # filter out whitespaces
    tTst = tLne.strip()
    tLen = len(tTst)
    if (tLen > 0):
        # count this line
        tCnt += 1

# display the metric results
tOut = "Physical SLOC: " + repr(tCnt) + " lines"
print tOut
```

Logical SLOC

This SLOC type counts only the *executable* lines of code in a given source (Listing 2). It ignores lines that contain only white spaces and those that contain comments. It still does not differentiate between branches and loops. To demonstrate, this code snippet has physical SLOC of *four* and a logical SLOC of *three*.

```
    /* This is a single-line comment */
    c = 0;
    for (i = 0; i <= 100; i++)
        c++;
```

Some logical SLOC metrics will divide lines that have two or more executable statements. This means the following snippet will have a logical SLOC of *three*, but a physical SLOC of *two*.

```
    /* This is a single-line comment */
    c = 0; for (i = 0; i <= 100; i++) c++;
```

Note the metric treats the three statements within the `'for(…)'` block as part of one executable line.

Finally, some logical SLOC metrics treat some statements as a single statement if they contain valid *continuation tokens*. For example, consider this AppleScript snippet.

```
    display dialog "This is a three line dialog statement" ¬
        buttons {"Cancel", "OK"} ¬
        with icon note
```

Here, the `'¬'` token links each line with the other. Thus, the above snippet has a physical SLOC of *three*, but a logical SLOC of *one*.

Listing 2. Counting executable lines.

```
#!/usr/bin/python
# import the following modules
import sys
import re

# read the data in STDIN
tInp = sys.stdin.readlines()

# count the number of executable code lines
tCnt = 0
tBlk = False
for tLne in tInp:
    # filter out whitespaces
    tTst = tLne.strip()
    tLen = len(tTst)
    if (tLen > 0):
        # filter out braces
        tFnd = re.search("^[ \t]*({|})", tTst)
        if (tFnd is None):
            # filter out inline comments
            tFnd = re.search("^[ \t]*//", tTst)
            if (tFnd is None):
                # filter out block comments
                # search for the opening token
                tFnd = re.search("^[ \t]*(/\*)", tTst)
                if (tFnd is None):
                    if (not tBlk):
                        # update the line count
                        tCnt += 1
                    else:
                        # search for closing token
                        tFnd = re.search("[ \t]*(\*/)", tTst)
                        if (tFnd is not None):
                            tBlk = False
                else:
                    # is this a single-line comment?
                    tFnd = re.search("[ \t]*(\*/)", tTst)
                    if (tFnd is None):
                        tBlk = True

# display the metric results
tOut = "Executable SLOC: " + repr(tCnt) + " lines"
print tOut
```

White-space SLOC

White-space SLOC (Listing 3) counts the *blank lines* that separate each executable and non-executable lines of code. White spaces affect the readability of a given source file. With the right amount of white space, we can distinguish each line of code and group lines of related code. Too much white space makes the code hard to read. Each line will be too far apart to be easily scanned. Too little makes the same code look crowded. Reading each line becomes arduous, especially within the restricted view of a source debugger.

Listing 3. Counting whitespace.

```
#!/usr/bin/python
# import the following modules
import sys
import re

# read the data in STDIN
tInp = sys.stdin.readlines()

# initialize the following locals
tPhy = 0
tWht = 0
```

```
# parse the code lines
for tLne in tInp:
    # filter out whitespaces
    tTst = tLne.strip()
    tLen = len(tTst)
    if (tLen > 0):
        # count the physical line
        tPhy += 1
    else:
        # count the whitespace
        tWht += 1


# display the metric results
tOut = "Physical SLOC: " + repr(tPhy) + " lines"
print tOut
tOut = "Whitespace SLOC: " + repr(tWht) + " lines"
print tOut
tPct = tWht * 100 / tPhy
tOut = "% SLOC (Whitespace/Physical): " + repr(tPct) + "
%"
print tOut
```

A good empirical rule is to keep a source's white-space SLOC within *10 to 20 percent* of its physical SLOC.

Comment SLOC

Another useful SLOC type is *comment SLOC* (Listing 4). This one counts the number of comments that are in a given source. Comments are essential to any well-maintained source. For without comments, we will be hard pressed in identifying a source's role or function. We will spend more time trying to trace its code than trying to improve or fix said code.

Listing 4. Counting comments.

```
#!/usr/bin/python
# import the following modules
import sys
import re

# read the data in STDIN
tInp = sys.stdin.readlines()

# initialize the following locals
tPhy = 0
tRem = 0
tBlk = False

# parse the code lines
for tLne in tInp:
    # filter out whitespaces
    tTst = tLne.strip()
    tLen = len(tTst)
    if (tLen > 0):
        # count the physical line
        tPhy += 1

        # check for block comments
        if (tBlk):
            # search for closing token
            tFnd = re.search("[ \t]*(\*/)", tTst)
            if (tFnd is not None):
                tBlk = False
            tRem += 1
        else:
            # check for inline comments
            tFnd = re.search("^.*//", tTst)
            if (tFnd is None):
                tFnd = re.search("^[ \t]*(/\*)[
\t\w]*(\*/)", tTst)
                if (tFnd is None):
                    # search for the opening token
                    tFnd = re.search("^[ \t]*(/\*)", tTst)
```

```
            if (tFnd is not None):
                tBlk = True
                tRem += 1
        else:
            tRem += 1
    else:
        tRem += 1

# display the metric results
tOut = "Physical SLOC: " + repr(tPhy) + " lines"
print tOut
tOut = "Comment SLOC: " + repr(tRem) + " lines"
print tOut
tPct = tRem * 100 / tPhy
tOut = "% SLOC (Comments/Physical): " + repr(tPct) + " %"
print tOut
```

Well-maintained sources should have *30 to 75 per cent* of its physical lines consisting of comments. Less than 30 per cent means the source is *poorly documented*. Greater than 75 per cent means the source is *mostly documentation*, with little to none of executable code. Yet, there are valid source files with a comment SLOC greater than 75 per cent. These files are often entry files, ones that have a `main()` routine.

Problems with SLOC

Of course, the SLOC metric is not without its issues and limits. For starters, SLOC is a poor measure of *coder output*. A well-skilled, experienced coder may use less lines of code than a neophyte coder to implement the same feature in a project. That same skilled coder may later use more lines of code to fix a bug in another project.

SLOC is grossly affected by the *source language*. Source files written in C may have a larger SLOC measure than those written in Object Basic. The use of external libraries also affect SLOC as they are often compiled, their sources not available.

SLOC is also a poor measure of *accountability*. In a team setup, there may be two or more coders working on the same file, each coder adding or changing code lines. Without knowing which lines belongs to which coder, SLOC will give the wrong measure for each coder.

The Halstead Metrics

Halstead metrics provide a more precise insight into code complexity. Proposed in 1977 by *Maurice H. Halstead* of Purdue University, Halstead metrics work by breaking down each code line into its constituent symbols. It then tallies the parts and uses the count results in seven empirical formulae. Like logical SLOC, Halstead metrics ignores lines that held only white space and comments. Nor does it check if a line is part of a branch or loop.

Operators and operands

To Halstead metrics, each executable line has two groups of symbols: *operators* and *operands*. Operators are symbols that define a *primitive language task*. They are the *unique* tokens and keywords that make up the source language. For example, assume the language is ANSI C. Its tokens include

BMS

THE LAW OFFICE OF
BRADLEY M. SNIDERMAN

Helping clients with their software legal issues.

- **Trademark and Copyright Registration**
- **Trade Secret Protection**
- **Licensing and Non Disclosure Agreements**
- **Assist with Software Audits**

I am an attorney practicing in Intellectual Property, Business Entity Formations, Corporate, Commercial and E-commerce Law.

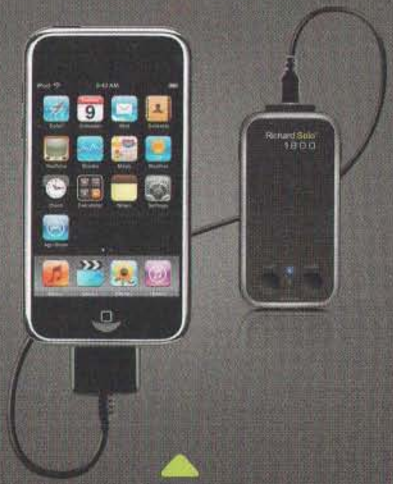
Please give me a call or an e-mail. Reasonable fees.

23679 Calabasas Rd. #558 • Calabasas, CA 91302
PHONE 818-706-0631 FAX 818-706-0651 EMAIL brad@sniderman.com

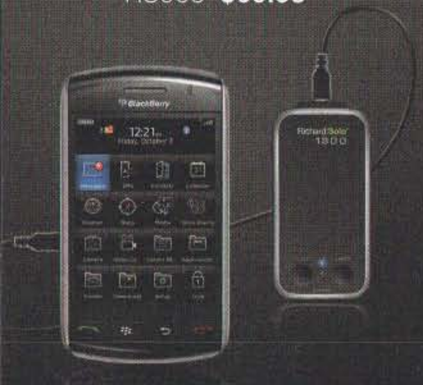
Built-in super bright LED flashlight.

Built-in laser pointer.

For iPhone 2G/3G/3GS/iPod
Direct plug-in / no cable.
Includes 2G / 3G support brace.
RS001 **$69.95**



For iPhone/iPod with Cable
Cable connection only.
Perfect for iPod touch.
RS008 **$69.95**



For All BlackBerry/Smartphones
Cable or direct plug-in.
Works with all USB-port phones.
RS007 **$69.95**

10 Reasons RichardSolo 1800 is the **best** backup battery for iPhone/ Smartphones — and *MacTech* saves you $20. **You pay only $49.95**

1. It is the only one that actually "latches" onto the iPhone — very stable.
2. Includes free, slim, protective hard case for iPhone 3G/3GS ($24.95 value) that works perfectly with included support brace.
3. Unlike "slipcase" configurations, there is no rear blockage of your cell phone antenna.
4. Licensed and certified by Apple for iPhone 2G/3G/3GS/iPod.
   Made for iPod Works with iPhone
5. Built-in flashlight is surprisingly useful and bright; laser pointer included.
6. Lightweight — you can easily carry it in your pocket, and top up your iPhone and iPod as needed.
7. Choose from two models: direct plug-in or cable. Both are 1800 mAh lithium-ion rechargeable!
8. We support you. Quick email response by the best customer service in the industry, and a full 1-year warranty.
9. **Free bonus** car charger and wall charger included. Charge battery and iPhone in tandem!
10. *MacTech* special price; enter the discount code ***MacTech*** at checkout. Instantly save $20!

Online ordering and blog reviews:
www.RichardSolo.com

Enter the discount code *MacTech* at checkout to instantly save $20. You pay only $49.95

Weekly GREAT DEAL Sign up today at RichardSolo.com to receive our weekly great deal email offer!

Order now online: www.RichardSolo.com

Richard | Solo®

the binary operators (`'+'`, `'-'`, `'/'`, `'*'`), unary operators (`'!'`, `'~'`) and logic operators (`'&&'`, `'||'`, `'^'`). Its keywords include the *primitive types* (`int`, `char`, `float`), the *loop symbols* (`do`, `for`, `while`) and the *branch symbols* (`if`, `switch`). Function calls do not qualify as operators for they can be broken down into operators and operands as well.

Operands are symbols that *convey data*. They include local and global variables, input arguments, string and number constants, functions, and routines. Operands are not unique to the source language, but they differ from one project to another.

Consider the following code snippet.

```
/* This is a single-line comment */
c = 0;
for (i = 0; i <= 100; i++)
    c++;
```

Line 2 of the snippet has two operators (`'='`, `';'`) and two operands (`'c'`, `'0'`). Line 3 has seven unique operators (`'for'`, `'='`, `';'`, `'<='`, `'++'`, `'('`, `')'`) and three unique operands (`'i'`, `'0'`, `'100'`). And line 4 has two operators (`'++'`, `';'`) and one operand (`'c'`). Since line 1 is a comment, the metric excludes that line. The metric also disregards the spaces that separate the operators and operands.

Dissecting the lines

Listing 5 shows the first part of the Xcode menu script implementing Halstead metrics. This part prepares the active source document to be analyzed. First, it uses the utility method `getTable()` to load the language symbols from two external files. The file `ansiC_keywords.txt` contains the reserved keywords of ANSI C. The file `ansiC_tokens.txt` contains its tokens. The script creates two dictionary objects, `tTkT` and `tKyT`, with which to hold the symbols. Next, the script reads the contents of the active source document. Then it calls its utility method `stripComments()`, which removes those lines that held only comments.

Listing 5. Preparing the source.

```
#!/usr/bin/python
# import the following modules
import sys
import re
import math

# define the following utility methods
# - Load the symbols table
def getTable(aNom):
    # load the symbols file
    tSrc = open(aNom, 'r')
    tTmp = tSrc.readlines()
    tSrc.close()

    # prepare the keyword table
    tTbl = {}
    for tWrd in tTmp:
        tKey = tWrd.strip()
        tTbl[tKey] = 0

    # return the symbols table
    return (tTbl)

# - Check for an inline/single-line comment
def isLineComment(aStr):
    # set the default result
    tChk = True

    # check for an inline comment
    tFnd = re.search("^.*//", aStr)
    if (tFnd is None):
        # check for a single-line comment
        tFnd =  re.search("^[ \t]*/\*[ \w]*\*/", aStr)
        if (tFnd is None):
            tChk = False

    # return the check result
    return (tChk)

# - Strip all code comments
def stripComments(aSrc):
    # initialize the following locals
    tOut = []
    tBlk = False

    # parse the original source lines
    for tLne in aSrc:
        # filter out whitespaces
        tTst = tLne.strip()
        tLen = len(tTst)
        if (tLen > 0):
            # check for an inline/single-line comment
            tChk = isLineComment(tTst)
            if not(tChk):
                # is this part of a block comment?
                if (tBlk):
                    # check for a closing token
                    tFnd = re.search("^[ \t]*(\*/)", tTst)
                    if (tFnd is not None):
                        tBlk = False
                else:
                    # check for the opening token
                    tFnd = re.search("^[ \t]*(/\*)", tTst)
                    if (tFnd is None):
                        # add the line to the output buffer
                        tOut.append(tTst)
                    else:
                        tBlk = True

    # return the output buffer
    return (tOut)

# - MAIN SCRIPT
# load the keyword table
tKyT = getTable('/Developer/Library/ansiC_keywords.txt')
tKyL = tKyT.keys()

# load the tokens table
tTkT = getTable('/Developer/Library/ansiC_tokens.txt')
tTkL = tTkT.keys()

# read the data in STDIN
tInp = sys.stdin.readlines()

# strip off all comments
tInp = stripComments(tInp)

#... CONTINUED ON LISTING 6
```

Listing 6 shows the second part of the menu script. This part divides each line into its constituent operators and operands. First, it prepares an empty dictionary `tOpT`, wherein it will store the operands. It makes a copy of each line and parses that copy for a token. Once it finds a token, the script updates the token count on `tTkT` and replaces the token with a *space*. It repeats the whole process until the line is empty of tokens.

Next, the script parses the stripped copy for keywords. Again, for each keyword it finds, the script updates the keyword count on **tKyT**. Then it replaces the keyword with a space. These same two steps repeat until the line is stripped of keywords. At this point, the line should consist *solely of operands and spaces*.

Listing 6. Parsing the symbols.

```
# import the following module
#... SEE LISTING 5

# initialize the operands table
tOpT = {}

# parse the source lines
for tLne in tInp:
    # copy the line
    tCpy = tLne

    # parse for tokens
    tEnd = False
    while not(tEnd):
        for tTkn in tTkL:
            tPos = tCpy.find(tTkn)
            if (tPos >= 0):
                # remove the token
                tCpy = tCpy.replace(tTkn, " ", 1)

                # update the token table
                tTkT[tTkn] += 1
                tEnd = False
                break
            else:
                tEnd = True

    # parse for keywords
    tEnd = False
    while not(tEnd):
        for tKwd in tKyL:
            tPos = tCpy.find(tKwd)
            if (tPos >= 0):
                # remove the token
                tCpy = tCpy.replace(tKwd, " ", 1)

                # update the token table
                tKyT[tKwd] += 1
                tEnd = False
                break
            else:
                tEnd = True

    # parse for operands
    tOpL = tCpy.split(' ')
    for tTmp in tOpL:
        tOpe = tTmp.strip()
        if (len(tOpe) > 0):
            if (tOpe in tOpT.keys()):
                tOpT[tOpe] += 1
            else:
                tOpT[tOpe] = 1

#... CONTINUED ON LISTING 7
```

Finally, the script uses the string function **split()** to separate each operand in the line. Each operand it finds gets added to the dictionary object **tOpT**. But if **tOpT** already has the operand, the script updates the operand count instead.

Calculating size

Halstead metrics defines two types of sizes. First, there is **code size** (Figure 6.a), or code length in some books. It is the total number of operators and operands in a given source. Next is **vocabulary size** (Figure 6.b). This is the total number of *unique* operators and operands in that same source. In ANSI C, the maximum vocabulary size is *75 symbols*. Each C source should have a vocabulary size *no greater* than the maximum.

$$N = N_{\text{operator}} + N_{\text{operand}}$$

(a) code size

$$\eta = \eta_{\text{operator}} + \eta_{\text{operand}}$$

(b) vocabulary size

Figure 6. Code and vocabulary sizes.

In Listing 7 is the part of the menu script that calculates the two sizes. First, the script scans its dictionary objects **tTkT** and **tKyT**. It sums up the number of tokens and keywords found in the active source. It also counts the tokens and keywords that appeared at least once in the source. Next, the script scans the dictionary object **tOpT**, and sums up the number of operands in the active source. Then it counts those operands that appeared at least once. Finally, the script adds up the counts and displays the resulting sizes—first the code size, then the vocabulary size.

Listing 7. Measuring sizes.

```
#... SEE LISTING 6
# count the number of operators
tNoT = 0
tNoU = 0
for tKey, tVal in tTkT.iteritems():
    if (tVal > 0):
        tNoT += tVal
        tNoU += 1

for tKey, tVal in tKyT.iteritems():
    if (tVal > 0):
        tNoT += tVal
        tNoU += 1

# count the number of operands
tNvT = 0
tNvU = 0
for tKey, tVal in tOpT.iteritems():
    if (tVal > 0):
        tNvT += tVal
        tNvU += 1

# — Halstead code size
tHmL = tNoT + tNvT
print "Halstead code size: %d symbols" % tHmL

# — Halstead vocabulary size
tHmS = tNoU + tNvU
print "Halstead vocabulary size: %d symbols" % tHmS

#... CONTINUED ON LISTING 8
```

Measuring volume

Another measure defined by Halstead metrics is **code volume** (Figure 7). Code volume is the *amount of information* held by a given project or source. It is expressed in terms of mathematical bits, and is less sensitive to code layout than logical SLOC.

$$V = N \cdot \log_2(\eta) = (N_{operator} + N_{operand}) \cdot \log_2(\eta_{operator} + \eta_{operand})$$

Figure 7. Code volume.

Listing 8 shows the portion of the Xcode menu script that computes code volume.

Listing 8. Computing code volume.

```
#... SEE LISTING 7
# — Halstead code volume
tHmV = tHmL * math.log(tHmS, 2)
print "Halstead code volume: %f bits" % tHmV

#... CONTINUED ON LISTING 9
```

A well-written source file has an average code volume *within 1000 to 3000 bits*. A routine, with or without inputs, usually has its code volume between *20 to 1000 bits*. If the routine has too large a volume, it is probably taking on *one too many tasks*. One way to alleviate this problem is to *refactor* the routine into two or more smaller routines.

Rating difficulty

In Halstead metrics, **difficulty** (Figure 8.a) is the effort required to *use* a source language. It changes in proportion to the number of unique operators in the source. Thus, a greater number of unique operators make a language harder to read. Difficulty also changes to the ratio of total operands and unique operands in the same language. If the language allows several instances of the same operands, the resulting source becomes harder to read and maintain.

$$D = \eta_{operator} \cdot \frac{N_{operand}}{2 \cdot \eta_{operand}}$$

(a) difficulty

$$L = \frac{1}{D}$$

(b) level

Figure 8. Language difficulty and level.

The inverse of difficulty is **language level** (Figure 8.b). Language level measures the *syntactical ease* of the source language. Low-level languages often have a large operator vocabulary and a limited range of operands. Good examples of such a language are, of course, assembly languages. High-level languages usually have a smaller set of operators, but they allow a greater range of operands. One example of a high-level language is Object Basic.

Listing 9 shows the part of the Xcode menu script that determines language difficulty.

Listing 9. Computing difficulty.

```
#... SEE LISTING 8
# — Halstead difficulty
tHmD = tNoU * tNvT * 1.0 / (2 * tNvU)
print "Halstead difficulty: %f " % tHmD
```

Asssessing effort

Halstead metrics consider **effort** as the amount of work needed *to read or maintain* a given source. This measure changes in proportion to both language difficulty and code volume (Figure 9.a). Thus, a source file takes more effort to maintain when it is written in assembly language. The same is also true if the source file carries a large amount of information.

$$E = D \cdot V$$

(a) effort

$$T_{implement} = \frac{E}{18} \text{ seconds}$$

(b) implementation time

$$B_{probable} = \frac{E^{\frac{2}{3}}}{3000}$$

(c) probable bugs

Figure 9. Estimating effort, time and bugs.

With effort established, Halstead metrics can make two other measures. The first measure (Figure 9.b) predicts the *time* needed to maintain the source file. But this measure must be calibrated first before use. Best way to do so is to apply Halstead metrics on a previous project, preferably one that is complete and has the same features as the current project.

The second measure (Figure 9.c) predicts the *number of bugs* that may appear in a given source file. Bug count usually varies with code complexity. Sources with mostly sequential lines of executable code have a lower bug count than those with lots of complex structures like branches, loop, and nests. Similarly, sources that span a couple of page are less likely to have bugs than those that take up five or more pages.

Listing 10 shows the part of the menu script that assesses effort, time and bugs.

Listing 10. Computing effort, time and bugs.

```
#... SEE LISTING 9
# — Halstead effort
tHmE = tHmD * tHmV

# — Halstead time
tHmT = tHmE * 1.0 / 18

# — Halstead bug estimate
tHmB = tHmE ** (2.0 / 3.0) / 3000.0

# display the metric results
print "Halstead effort: %f " % tHmE
print "Halstead time: %f seconds" % tHmT
print "Halstead bug estimate: %f bugs" % tHmB
```

Be a hero.

CRASHPLANPRO™
Continuous Backup for Business

Protect everyone, everywhere.

Only CrashPlan protects your workforce everywhere with real-time onsite and cloud backup.

Peace, justice and backup for all.

crashplanpro.com/hero

A good rule of thumb is limit the probable bug count to *two*. Branches and loops must be kept as simple as possible, and nested structures must be kept to a minimum. If the source file is too large, it may help to refactor the file into several smaller files.

Summing Up

In this article, we examined two common types of source metrics. The first metrics, SLOC, quantifies a given project or source in lines of code. It separates those lines that are executable from those that are essentially descriptive. Then it uses these line counts to assess the project or its sources.

The second metric, Halstead, breaks down the source into its operators and operands. It counts the number of times an operator or operand appears. Then it uses those counts to estimate language difficulty, effort, implementation time, and probable bugs. Unlike SLOC, Halstead ignores all comments and white space.

We also caught a glimpse of the new Xcode script editor. We learned how to use the editor to manage Xcode's menu script and to add a new script. We also learned how to control the flow of data to and from the script.

Source metrics play a beneficial role in project management. When used properly, source metrics can help assess project growth and locate potential problems. We have but glimpsed the potential of a source metrics. In a later article, we will explore other useful metrics and learn how to implement them as an Xcode menu script.

Until then, I bid you well.

Bibliography and References

Maurice H. Halstead. "The Essential Design Criterion for Computer Languages: Software Science." Computer Science Department, TR191. Purdue University.

James Peters and Witold Pedrycz. "Software Engineering—An Engineering Approach." New York:John Wiley & Sons, Inc. 1998.

VerifySoft. "Measurement of Halstead Metrics with Testwell CMT++ and CMT Java (Complexity Measures Tool)." VerifySoft Technology. Internet: http://www.verifysoft.com/en_halstead_metrics.html. [2010 Apr 04].

VerifySoft. "Measurement of Lines of Code Metrics with Testwell CMT++ and CMT Java (Complexity Measures Tool)." VerifySoft Technology. Internet: http://www.verifysoft.com/en_linesofcode_metrics.html. [2010 Apr 26].

Wikipaedia. "Halstead Complexity Measures." The Wikipaedia Community, 2009 Oct 15. Available: http://en.wikipedia.org/wiki/Halstead_complexity_measu res. [2010 Apr 04].

Wikipaedia. "Source Lines of Code." The Wikipaedia Community, 2010 Apr 04. Available: http://en.wikipedia.org/wiki/Source_lines_of_code. [2010 Apr 04].

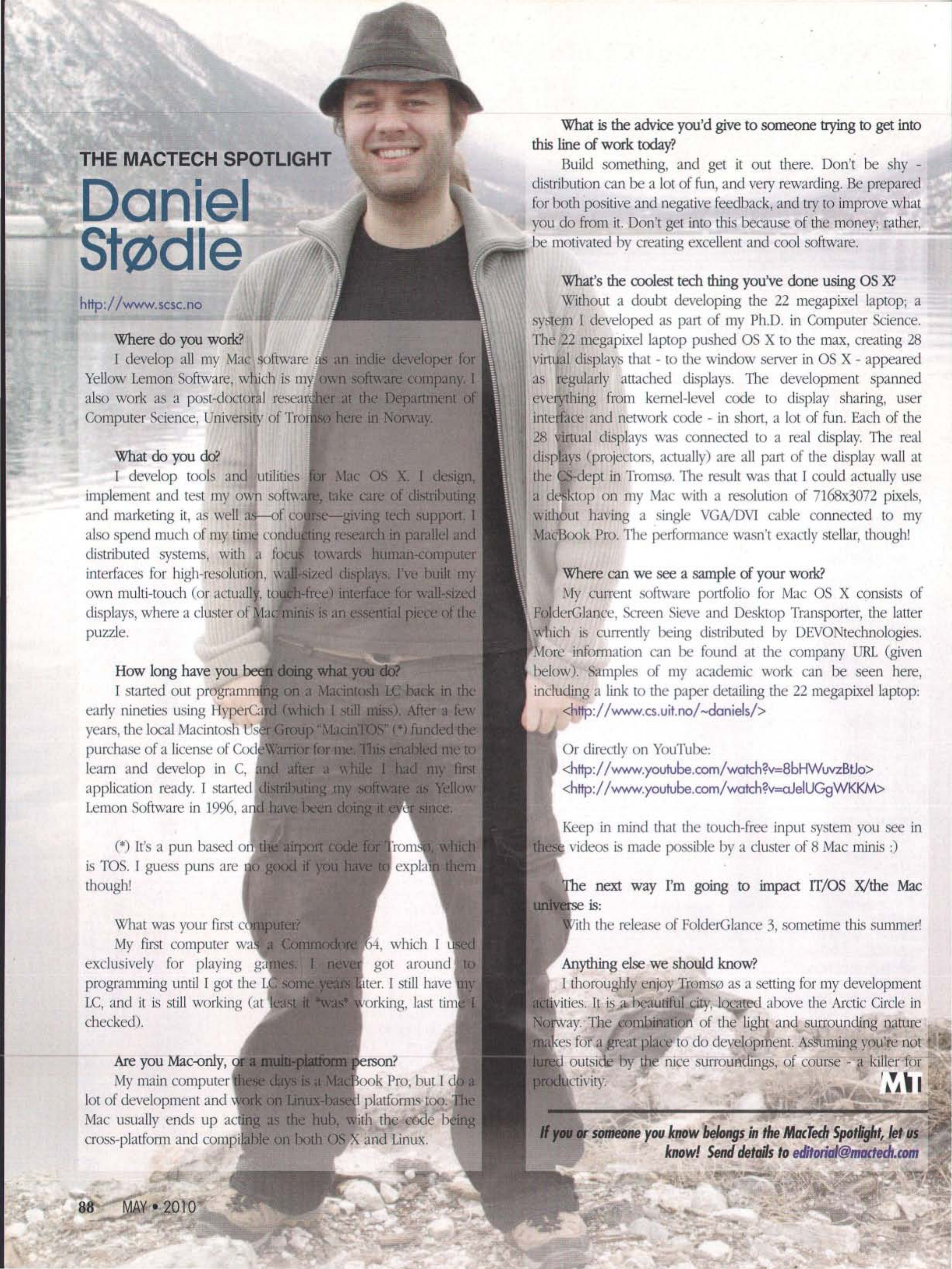
Horst Zuse. "Resolving the Mysteries of the Halstead Measures." VerifySoft Technology. [Online]. Available: http://www.horst-zuse.homepage.t-online.de/z-halstead-final-05-1.pdf. [2010 Apr 04].

MT

About The Author

JC is a freelance engineering writer from North Vancouver, British Columbia. He writes for various publications, covering diverse topics such as AppleScript, REALbasic, Python, and Cocoa. He also spends quality time with his foster nephew. You can reach JC at anarakisware@gmail.com.

Advertiser/Product Index



THE MACTECH SPOTLIGHT

Daniel Stødle

http://www.scsc.no

Where do you work?

I develop all my Mac software as an indie developer for Yellow Lemon Software, which is my own software company. I also work as a post-doctoral researcher at the Department of Computer Science, University of Tromsø here in Norway.

What do you do?

I develop tools and utilities for Mac OS X. I design, implement and test my own software, take care of distributing and marketing it, as well as—of course—giving tech support. I also spend much of my time conducting research in parallel and distributed systems, with a focus towards human-computer interfaces for high-resolution, wall-sized displays. I've built my own multi-touch (or actually, touch-free) interface for wall-sized displays, where a cluster of Mac minis is an essential piece of the puzzle.

How long have you been doing what you do?

I started out programming on a Macintosh LC back in the early nineties using HyperCard (which I still miss). After a few years, the local Macintosh User Group "MacinTOS" (*) funded the purchase of a license of CodeWarrior for me. This enabled me to learn and develop in C, and after a while I had my first application ready. I started distributing my software as Yellow Lemon Software in 1996, and have been doing it ever since.

(*) It's a pun based on the airport code for Tromsø, which is TOS. I guess puns are no good if you have to explain them though!

What was your first computer?

My first computer was a Commodore 64, which I used exclusively for playing games. I never got around to programming until I got the LC some years later. I still have my LC, and it is still working (at least it *was* working, last time I checked).

Are you Mac-only, or a multi-platform person?

My main computer these days is a MacBook Pro, but I do a lot of development and work on Linux-based platforms too. The Mac usually ends up acting as the hub, with the code being cross-platform and compilable on both OS X and Linux.

What is the advice you'd give to someone trying to get into this line of work today?

Build something, and get it out there. Don't be shy - distribution can be a lot of fun, and very rewarding. Be prepared for both positive and negative feedback, and try to improve what you do from it. Don't get into this because of the money; rather, be motivated by creating excellent and cool software.

What's the coolest tech thing you've done using OS X?

Without a doubt developing the 22 megapixel laptop; a system I developed as part of my Ph.D. in Computer Science. The 22 megapixel laptop pushed OS X to the max, creating 28 virtual displays that - to the window server in OS X - appeared as regularly attached displays. The development spanned everything from kernel-level code to display sharing, user interface and network code - in short, a lot of fun. Each of the 28 virtual displays was connected to a real display. The real displays (projectors, actually) are all part of the display wall at the CS-dept in Tromsø. The result was that I could actually use a desktop on my Mac with a resolution of 7168x3072 pixels, without having a single VGA/DVI cable connected to my MacBook Pro. The performance wasn't exactly stellar, though!

Where can we see a sample of your work?

My current software portfolio for Mac OS X consists of FolderGlance, Screen Sieve and Desktop Transporter, the latter which is currently being distributed by DEVONtechnologies. More information can be found at the company URL (given below). Samples of my academic work can be seen here, including a link to the paper detailing the 22 megapixel laptop:

<http://www.cs.uit.no/~daniels/>

Or directly on YouTube:

<http://www.youtube.com/watch?v=8bHWuvzBtJo>

<http://www.youtube.com/watch?v=aJelUGgWKKM>

Keep in mind that the touch-free input system you see in these videos is made possible by a cluster of 8 Mac minis :)

The next way I'm going to impact IT/OS X/the Mac universe is:

With the release of FolderGlance 3, sometime this summer!

Anything else we should know?

I thoroughly enjoy Tromsø as a setting for my development activities. It is a beautiful city, located above the Arctic Circle in Norway. The combination of the light and surrounding nature makes for a great place to do development. Assuming you're not lured outside by the nice surroundings, of course - a killer for productivity. MT

If you or someone you know belongs in the MacTech Spotlight, let us know! Send details to editorial@mactech.com

Mac shopping made easy.

Grab that to-do list, and prepare for some one-stop shopping at Smalldog.com!

Bundles simplify the buying process

Mac bundles (think Mac + RAM + AppleCare + external hard drive, etc.) not only include **everything you need,** but also **save you money.**

Visit » **Smalldog.com/specials**

Macs from under $500

We carry all **current Macs** as well as **used, refurbished and closeout models,** so there is a Mac for any budget.

Visit » **Smalldog.com/macs**

Free shipping over $200

It's true–we provide **free, same-day ground shipping** on every item over $200 every day.

Tax-free shopping

Purchases outside of Vermont are always shipped **tax-free.**

Small Dog Electronics
Always By Your Side

www.smalldog.com

800-511-MACS

 Apple Specialist

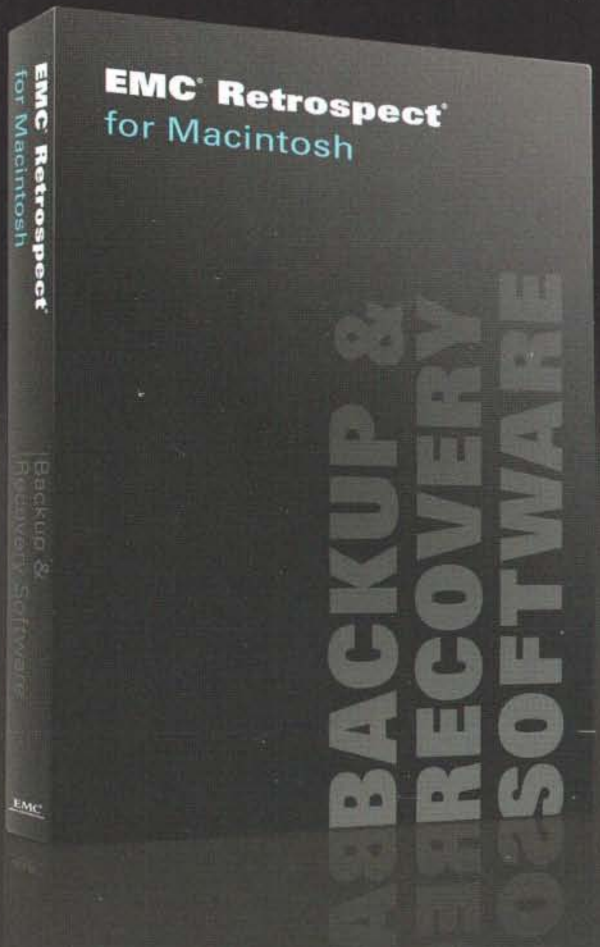
Celebrating 15 Years • 3rd Largest Apple Specialist in New England • 5-Star Merchant Rating • Same-day shipping

Bundles
Macs
Free Shipping
Tax-Free

NEW!

EMC® Retrospect® 8

backup and recovery software for small and medium businesses


EMC Retrospect
for Macintosh
BACKUP & RECOVERY SOFTWARE
EMC Retrospect for Macintosh
Backup & Recovery Software
EMC

The most trusted name in Mac backup

All-new EMC Retrospect 8 for Macintosh provides the reliability, ease of use, power, and flexibility you need to protect critical data on Mac and Windows PCs and servers. EMC Retrospect includes a state-of-the-art Mac user interface and enterprise-level features — including remote management of one or more backup servers, disk-to-disk-to-*anything* backups, Xsan support and custom reporting — at a fraction of the cost of other products.

Download a free 45-day trial at **www.retrospect.com/wwdc**

EMC² where information lives®